NOTE: The cover page of this standard has been changed for administrative reasons. There are no other changes to this document.

NOT MEASUREMENT SENSITIVE

MIL-STD-2045-44500
18 June 1993

DEPARTMENT OF DEFENSE
INTERFACE STANDARD

TACTICAL COMMUNICATIONS PROTOCOL 2
(TACO2)
FOR THE
NATIONAL IMAGERY TRANSMISSION FORMAT STANDARD

AMSC N/A

AREA DCPS

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

FOREWORD

1. The National Imagery Transmission Format Standards (NITFS) is the standard for formatting digital imagery and imagery-related products and exchanging them among members of the Intelligence Community (IC) as defined by Executive Order 12333, the Department of Defense (DOD), and other departments and agencies of the United States Government as governed by Memoranda of Agreement (MOA) with those departments and agencies.

2. The National Imagery Transmission Format Standards Technical Board (NTB) developed this standard based upon currently available technical information.

3. The DOD and members of the Intelligence Community are committed to interoperability of systems used for formatting, transmitting, receiving, and processing imagery and imagery-related information. This standard describes the TActical COmmunication protocol 2 (TACO2) requirements and establishes its application within the NITFS.

4. As a result of a Defense Information Systems Agency (DISA) action, standards for all military data communication protocols will be published in a MIL-STD-2045 series of documents. A MIL-STD-2045 document series has been established within the Data Communications Protocol Standards (DCPS) standardization area.

a. MIL-STD-2045-10000 series. MIL-STD-2045-10000 to MIL-STD-2045-19999 inclusive, will be used to describe DOD's implementation of commercial, international, national, federal, and military standards within the functional profile concept, in order to provide required network services. U.S. Government Open Systems Interconnection Profile (GOSIP) will be the basis for developing the 10000 series with DOD enhancements and unique military standards.

b. MIL-STD-2045-20000 series. MIL-STD-2045-20000 to MIL-STD-2045-29999 inclusive, will be used to describe DOD enhancements and extensions to existing commercial, international, national, or federal standards.

c. MIL-STD-2045-30000 series. MIL-STD-2045-30000 to MIL-STD-2045-39999 inclusive, will be used to describe DOD unique protocols and services that are not supported by commercial, international, national, or federal standards.

d. MIL-STD-2045-40000 series. MIL-STD-2045-40000 to MIL-STD-2045-49999 inclusive, will be used to document interim standards. Interim standards are documents DOD needs until these standards are described in either GOSIP or MIL-STD-2045-20000 or 30000 series standards.

5. Beneficial comments (recommendations, additions, deletions) and any pertinent data which may be of use in improving this document should be addressed to Defense Information Systems Agency (DISA), Joint Interoperability and Engineering Organization (JIEO), Center for Standards (CFS), Attn: TBBD, Fort Monmouth, NJ 07703-5613 by using the Standardization Document Improvement Proposal (DD Form 1426) appearing at the end of this document or by letter.

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

PARAGRAPH PAGE

CONTENTS

PARAGRAPH PAGE

CONTENTS

PARAGRAPH PAGE

FIGURE

CONTENTS

PARAGRAPH PAGE

TABLE

APPENDIX

1. SCOPE

1.1 Scope. This document establishes the requirements for the TActical COmmunications protocol 2 (TACO2), part of the National Imagery Transmission Format Standards (NITFS). National Imagery Transmission Format (NITF) is a standard format for transmitting digital imagery and imagery-related products among members of the Intelligence Community, and TACO2 is a protocol suite that may be used for that transmission. It includes requirements for Forward Error Correction (FEC), which is necessary to ensure interoperability and to promote commonality among subsystems that comply with NITFS.

1.2 Content. This standard establishes the requirements to be met by systems complying with NITFS when using the TACO2 protocol, and defines the protocols and formats that make up TACO2. All aspects of TACO2 that affect functional interoperability are specified herein. In addition, guidance is provided for those aspects of TACO2 operation that are not strictly related to interoperability but may affect technical performance or resistance to error.

1.3 Applicability. This standard is applicable to the Intelligence Community and the DOD. It is mandatory for all Secondary Imagery Dissemination Systems (SIDS) in accordance with the memorandum by the Assistant Secretary of Defense for Command, Control, Communications, and Intelligence ASD(C^3I), Subject: National Imagery Transmission Format Standards (NITFS), 12 August 1991. This directive shall be implemented in accordance with JIEO Circular 9008, and MIL-HDBK-1300. New equipment and systems, those undergoing major modification, or those capable of rehabilitation shall conform to this standard.

1.4 Protocol tailoring. TACO2 is designed as a single protocol stack that provides for message transfer over a wide variety of tactical communication circuits. It is particularly appropriate for use over circuits where other protocol suites operate poorly or not at all, but also is designed to perform well over any communications circuit. It can transfer any form of data, since it does not use any internal component of an NITFS message. It can be configured to operate over circuits not anticipated at initial installation; therefore, a conforming TACO2 implementation must implement all capabilities specified herein, except as specifically noted. The possible ranges of various parameters may be limited for specific applications; mandatory ranges are specified in this document. Additional information on NITFS compliance is available in JIEO Circular 9008.

1.5 FEC tailoring. As a minimum, only those features or functions specified herein, necessary to ensure interoperability among systems, shall be implemented in an equipment item. While every effort has been made to include all the features necessary, certain aspects depend on system application and must be tailored by the specification writer. These aspects include:

a. User choice of appropriate FEC selection.

b. Automatic switching of FEC code based on the conditions of the tactical line.

c. Inhibiting external or internal FEC codes.

d. Using an external FEC code if it is desired.

2. APPLICABLE DOCUMENTS

2.1 Government documents.

2.1.1 Specifications, standards, and handbooks. The following specifications, standards, and handbooks form a part of this document to the extent specified herein. Unless otherwise specified, the issues of these documents are those listed in the issue of the Department of Defense Index of Specifications and Standards (DODISS) and supplements thereto, cited in the solicitation.

STANDARDS

FEDERAL

FED-STD-1037B	-	Telecommunications: Glossary of Telecommunication Terms, 3 June 1991.

MILITARY

MIL-STD	-	1777Military Standard Internet Protocol, Defense Communications Agency, August 1983.
MIL-STD-188-114A	-	Electrical Characteristics of Digital Interface Circuits, 30 September 1985.
MIL-STD-2500	-	National Imagery Transmission Format (NITF) for the National Imagery Transmission Format Standards (NITFS), 18 June 1993.

HANDBOOKS

MIL-HDBK-1300	-	National Imagery Transmission Format Standards (NITFS), 18 June 1993.

(Unless otherwise indicated, copies of federal and military specifications, standards, and handbooks are available from the Standardization Documents Order Desk, 700 Robbins Avenue, Building #4, Section D, Philadelphia, PA 19111-5094.)

2.1.2 Other Government documents, drawings, and publications. The following other Government documents, drawings, and publications form a part of this document to the extent specified. Unless otherwise specified, the issues are those cited in the solicitation.

DISA/JIEO Circular 9008	-	NITFS Certification Test and Evaluation Plan, (Effectivity 8).
DISA/JIEO SPEC 9137	-	NITFS TACO2 Protocol to KY-57/58 Cryptographic Device Technical Interface Specification (TIS), (Effecitivity 8).
DISA/JIEO SPEC 9138	-	NITFS TACO2 Protocol to KG-84-A/C Cryptographic Device Technical Interface Specification (TIS), (Effectivity 8).
DISA/JIEO SPEC 9139	-	NITFS TACO2 Protocol to KY-68 Cryptographic Device Technical Interface Specification (TIS), (Effectivity 8).
DISA/JIEO SPEC 9140	-	NITFS TACO2 Protocol to STU-III Cryptographic Device Technical Interface Specification (TIS), (Effectivity 8).

(Copies of DISA/JIEO Specifications may be obtained from DISA/JIEO/CFS/TBB, Fort Monmouth, NJ 07703-5613. Copies of DISA/JIEO Circular 9008 may be obtained from DISA/JIEO/JITC, Fort Huachuca, AZ 85613-7020.)

2.2 Non-Government publications. The following document(s) form a part of this document to the extent specified herein. Unless otherwise specified, the issues of the documents which are DOD adopted are those listed in the issue of the DODISS cited in the solicitation. Unless otherwise specified, the issues of documents not listed in the DODISS are the issues of the documents cited in the solicitation.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION

ISO 3309	-	High-Level Data Link Control Procedures - Frame Structure, International Organization for Standardization, Switzerland, 15 January 1992.
ISO 7498	-	Open systems interconnection - basic reference model International Organization for Standardization, Switzerland.
ISO 8825	-	Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), International Organization for Standardization, Switzerland, 15 December 1990.

ISO 9171	-	Recorded/Unrecorded Characteristics of 130 mm Optical Disk Cartridges.

AMERICAN NATIONAL STANDARDS INSTITUTE (ANSI)

ANSI X3.4-198	-	American National Standard Code for Information Interchange (ASCII), 1986.

INTERNET RFCs

RFC 791	-	Internet Protocol, Postel, J.B., 1981.
RFC 792	-	Internet Control Message Protocol, Postel, J.B., 1981.
RFC 919	-	Broadcasting Internet datagrams, Mogul, J.C., 1984.
RFC 922	-	Broadcasting Internet datagrams in the presence of subnets, Mogul, J.C., 1984.
RFC 950	-	Internet standard subnetting procedure, Mogul, J.C.; Postel, J.B., 1985.
RFC 998	-	NETBLT: A bulk data transfer protocol, Clark, D.D.; Lambert, M.L.; Zhang, L., 1987.
RFC 1055	-	Nonstandard for transmission of IP datagrams over serial lines: SLIP, Romkey, J.L., 1988.
RFC 1108	-	Security Options for the Internet Protocol, Kent, S., 1991.
RFC 1112	-	Host extensions for IP multicasting, Deering, S.E., 1989.

(Non-Government standards and publications are usually available from the organization that prepare or distribute the documents. These documents also may be available in or through libraries or other informational services.)

2.3 Order of precedence. In the event of a conflict between the text of this standard and the references cited herein, the text of this standard takes precedence. Nothing in this document, however, supersedes applicable laws and regulations unless a specific exemption has been obtained.

3. DEFINITIONS

3.1 Acronyms used in this standard. The following definitions are applicable for the purpose of this standard. In addition, terms used in this standard and defined in the FED-STD-1037B shall use the FED-STD-1037B definition unless noted.

a.	ANSI	American National Standards Institute
b.	ASCII	American Standard Code for Information Interchange
c.	ASD(C^3I)	Assistant Secretary of Defense for Command, Control, Communications, and Intelligence
d.	ASN.1	Abstract Syntax Notation One
e.	BCH	Bose-Chaudhuri-Hocquenghem
f.	BER	Bit Error Ratio
g.	BERT	Bit Error Ratio Test
h.	BPSK	Binary Phase Shift Keying
i.	CCITT	International Telegraph and Telephone Consultative Committee
j.	CFS	Center for Standards
k.	CRC	Cyclic Redundancy Check
l.	DAMA	Demand Assignment Multiple Access
m.	DCE	Data Circuit-terminating Equipment
n.	DCPS	Data Communications Protocol Standards
o.	DDN	Defense Data Network
p.	DISA	Defense Information Systems Agency (formerly DCA)
q.	DISN	Defense Information Systems Network (formerly DDN)
r.	DOD	Department of Defense
s.	DODISS	Department of Defense Index Specifications and Standards

t.	DTE	Data Terminal Equipment
u.	EDAC	Error Detection and Correction
v.	EDAC	Error Detection and Correction
w.	FCS	Frame Check Sequence
x.	FEC	Forward Error Correction
y.	FTP	File Transfer Protocol
z.	GOSIP	U.S. Government OSI Profile
aa.	HDLC	High-level Data Link Control
ab.	HF	High Frequency
ac.	IC	Intelligence Community
ad.	ICMP	Internet Control Message Protocol
ae.	IHL	Internet Header Length
af.	IP	Internet Protocol
ag.	ISO	International Organization for Standardization
ah.	JIEO	Joint Interoperability and Engineering Organization (formerly JTC3A)
ai.	LSB	Least Significant Bit
aj.	LOS	Line of Sight
ak.	MBZ	Must Be Zero
al.	MOA	Memoranda of Agreement
am.	MSB	Most Significant Bit
an.	msec	Milliseconds
ao.	NETBLT	NETwork BLock Transfer

ap.	NITF	National Imagery Transmission Format
aq.	NITFS	National Imagery Transmission Format Standards
ar.	NRTS	National Imagery Transmission Format Reliable Transfer Server
as.	NTB	National Imagery Transmission Format Standard Technical Board
at.	OSI	Open Systems Interconnection
au.	RFC	Request for Comment (Internet environment)
av.	SID	Secondary Imagery Dissemination
aw.	SIDS	Secondary Imagery Dissemination System
ax.	SLIP	Serial Line Internet Protocol
ay.	TACO2	TActical COmmunications protocol 2
az.	TBR	To Be Resolved
ba.	TCP	Transmission Control Protocol
bb.	TIS	Technical Interface Specification
bc.	TRI-TAC	Tri-Service Tactical Communications
bd.	UDP	User Datagram Protocol
be.	UHF	Ultra High Frequency
bf.	UID	Unique Identifier
bg.	VHF	Very High Frequency
bh.	UI	Unnumbered Information

3.2 Definitions used in this standard. The definitions used in this document are defined as follows:

a. Bit error ratio test (BERT) - A function or sequence of functions that compares a received data pattern with a known transmitted pattern to determine the level of transmission quality. Note: Can be used as an adjective, for example, "Bit error ratio test packets" are packets used in a bit error ratio test.

b. Bit-stuffing - For NITFS, in High-level Data Link Control (HDLC), a technique used to avoid spurious appearances of the flag within a frame.

c. Bose-Chaudhuri-Hocquenchem (BCH) codes - An important class of binary, block forward error correction (FEC) codes. BCH codes offer a great deal of flexibility in terms of code rate and block length. Hamming codes may be thought of as single error-correcting BCH codes.

d. Byte - A sequence of N adjacent binary digits, usually treated as a unit, where N is a non zero integral number. Note: In pre-1970 literature, "byte" referred to a variable length field. Since that time the usage has changed so that now it almost always refers to an 8-bit field. This usage predominates in computer and data transmission literature; throughout this document, the term is synonymous with "octet."

e. Byte-stuffing - A procedure in which either the Huffman coder or the arithmetic coder inserts a zero byte into the entropy-coded segment following the generation of an encoded hexadecimal 0xFF byte. For the purpose of NITFS, in Serial Line Internet Protocol (SLIP), a technique used to avoid spurious appearances of the END character within a frame.

f. Client - An executing program or protocol layer that requests or receives services from a lower protocol layer.

g. Criticality - Those portions of a message which must be received correctly for the message to be useful are considered critical. Criticality provides a means for identifying those portions of a message.

h. Datagram - In packet-switching, a self-contained packet, independent of other packets, that carries information sufficient for routing from the originating data terminal equipment to the destination data terminal equipment, without relying on earlier exchanges between the equipment and the network. Note: Unlike virtual call service, there are no call establishment or clearing procedures, and the network does not generally provide protection against loss, duplication, or misdelivery.

i. Data link layer - Layer two in the ISO OSI Reference Model. The role of the data link layer is to group the bits of the physical layer into frames, and to deal with transmission errors to allow the sending of frames between adjacent nodes in the network.

j. Duplex - For the purpose of this MIL-STD, an operational mode in which frames may be transferred across a link in both directions; i. e., half-duplex or full-duplex.

k. Effectivity - Some of the capabilities specified in this document are not required as of the issue date of the document. All such capabilities are marked with effectivity numbers, for example, (Effectivity 1). Each effectivity number will be replaced by a specific date in subsequent releases of this document.

l. Embedded FEC - For the purpose of this MIL-STD, FEC is an element of a hardware unit with more general functionality.

m. Error Detection and Correction (EDAC) - The application of one or several methods for the detection and correction of errors in a bit stream. For the purpose of this MIL-STD, EDAC generally is used synonymously with FEC, but is sometimes used to refer to error control systems that make use of a backward channel (for example, retransmission requests).

n. Finite field - See Galois field.

o. Forward Error Correction (FEC) - A system of error control for data code transmission wherein the receiving device has the capability to detect and correct any character or block that contains fewer than a predetermined number of symbols in error. Note: FEC is accomplished by adding bits to each transmitted character or code block using a predetermined algorithm.

p. Frame - 1. For the purpose of this MIL-STD, in data transmission, a sequence of contiguous bits bracketed by and including uniquely recognizable delimiters. 2. For the MIL-STD-188-198 (JPEG), a group of one or more scans (all using the same DCT-based or lossless process) through the data of one or more of an image.

q. Full duplex - For the purpose of this MIL-STD, an operational mode in which frames may be simultaneously transferred across a link in both directions. A TACO2 connection supports image transmission in only one direction at a time; return frames contain control information only.

r. Galois field - An algebraic structure commonly used for error correction and cryptographics calculations. A Galois field is a field whose set of elements is finite. The field operations of addition, subtraction, multiplication, and division are defined.

s. International Organization for Standardization (ISO) - A global standards body.

t. ISO OSI Reference Model - A seven layer protocol stack defined by the ISO.

u. Keepalive - A signal whose purpose is to inform a process that a connection is still in operation.

v. Metamessage - A collection of information related to a NITF message, which is transmitted in association with the message.

w. Modem - Acronym for Modulator-Demodulator. A device that modulates and demodulates signals. Note: 1. Modems are primarily used for converting digital signals into quasi-analog signals for transmission over analog communication channels and for reconverting the quasi-analog signals into digital signals. Note: 2. Many additional functions may be added to a modem to provide for customer service and control features.

x. Multicast - Transmission of a single message to a group of receivers.

y. Network layer - Layer three in the ISO OSI Reference Model. The role of the network layer is to transfer packets from their source node to their destination node by hopping through the intermediate nodes.

z. Octet - A byte of eight binary digits usually operated upon as an entity.

aa. Packet - In data communication, a sequence of binary digits, including data and control signals, that is transmitted and switched as a composite whole. The data, control signals, and possibly error control information are arranged in a specific format.

ab. Physical layer - Layer one in the ISO OSI Reference Model. The role of the physical layer is that of raw transmission of unformatted information.

ac. Port - For the NITFS, the identifier that transport protocols use to distinguish among multiple destinations in a host computer.

ad. Protocol stack - A set of multiple layers that describe the function of a network or communication system with the uppermost layer is being associated with the application and the lowest layer's being associated with the physical communications channel.

ae. Reed-Solomon code - For the purpose of NITF, a class of FEC codes in which the input and output symbols are multi-bit symbols and are treated as Finite Field elements.

af. Simplex - For NITFS, providing transmission in only one preassigned direction.

ag. Validity - Validity provides a means of identifying those portions of a message known to contain possible errors.

4. GENERAL REQUIREMENTS

4.1 Overview. To exchange NITFS messages between systems, the participants must agree on the mechanism and protocols to be used to support the exchange. In some cases, connectivity and transfer protocols already may exist; for instance, Defense Information Systems Network (DISN)-connected hosts can use File Transfer Protocol (FTP) for moving standard format files. In other cases, connectivity is available, but common transfer protocols are not, or the available protocols are intolerably inefficient; for instance, DISN protocols run very slowly over slow-turnaround half-duplex circuits, and cannot run at all over simplex circuits. TACO2 provides efficient NITFS message transfer across point-to-point and point-to-multipoint links (tactical radio circuits) where neither DISN nor other current GOSIP protocols are suitable.

4.1.1 Approach. TACO2 uses a layered model, similar in philosophy to the ISO Open Systems Interconnection Reference Model. The TACO2 model is shown on figure 1; the components are described in the following subparagraphs.

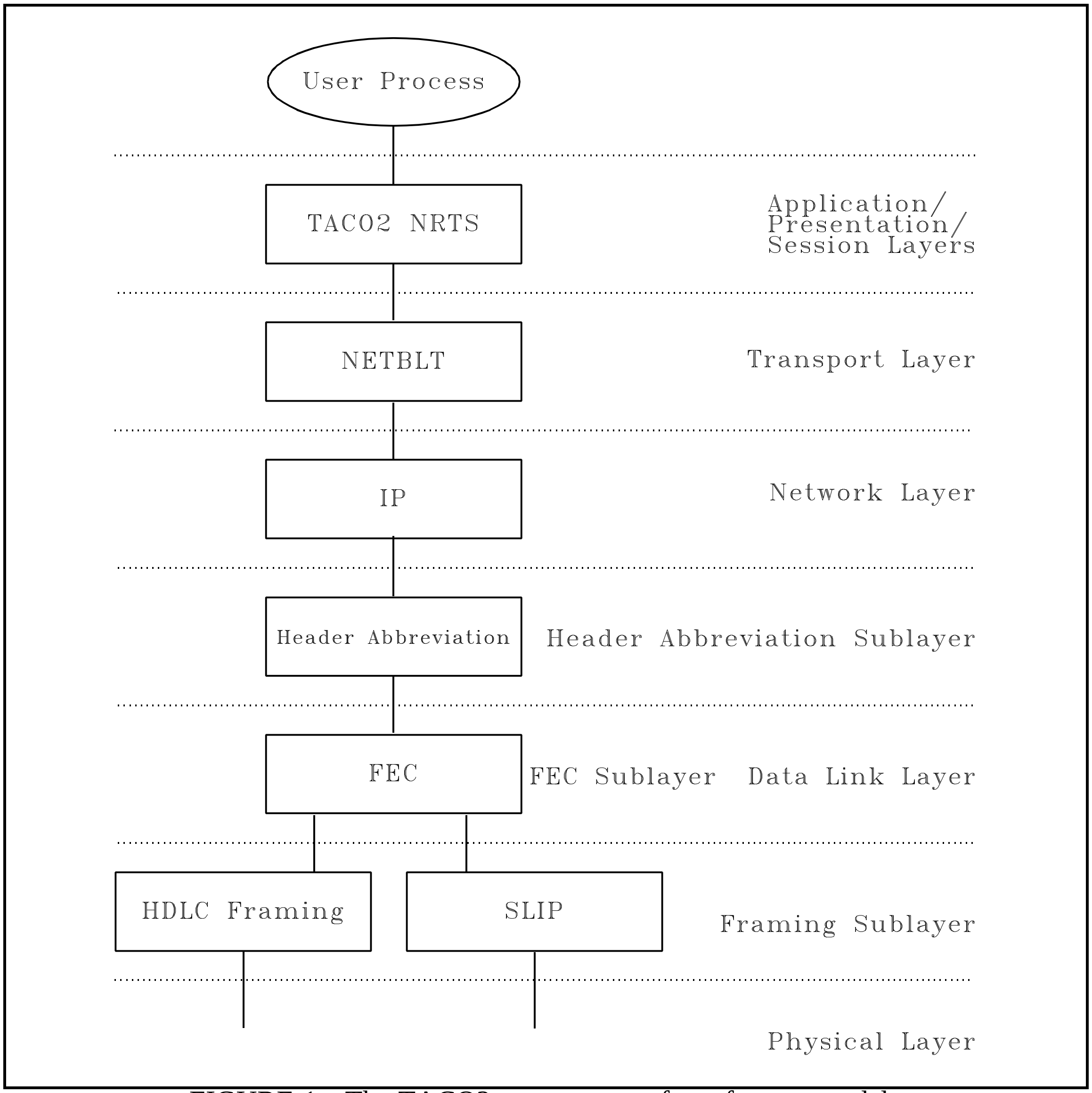

FIGURE 1. The TACO2 message transfer reference model.

4.1.2 NITFS reliable transfer server for TACO2 (TACO2 NRTS). The TACO2 NRTS controls the communications service to be used, exchanges the message and associated information with it, and acts as a session layer to allow resumption of interrupted transfers.

4.1.3 NETBLT. NETwork BLock Transfer (NETBLT) shall provide the reliable, flow-controlled transport level protocol designed to achieve high throughput across a wide variety of networks. It allows the sending client (the NRTS) to break the message being sent into a series of buffers, and to pass those buffers to NETBLT as information or space availability permits. Buffers are broken into packets for transmission. The critical element for performance is multiple buffering, so that new buffers can be sent while earlier ones await confirmation, and packet flow can be nearly continuous from sender to receiver. NETBLT uses a system of timers, acknowledgments, and recovery mechanisms to deal with network congestion, long delays, and errors. NETBLT can operate in full-duplex or half-duplex modes. The simplex case uses a stripped-down version of the same NETBLT protocol.

4.1.4 IP. For the next lower layer, TACO2 shall use the connectionless, unreliable datagram delivery Internet Protocol (IP), which includes the Internet Control Message Protocol (ICMP).

4.1.5 Header Abbreviation sublayer. TACO2 provides a mechanism for header abbreviation across point-to-point links. Using the header abbreviation sublayer is optional; its inclusion in any compliant implementation of TACO2 shall be mandatory (Effectivity 3).

4.1.6 FEC. FEC is a mandatory component of the TACO2 protocol stack whose use in a particular circuit is user selectable (Effectivity 1). Two coding algorithms are defined: one is a Reed-Solomon code for operation in moderate error environments, the other is a combination of Reed-Solomon and Bose-Chaudhuri-Hocquenghem (BCH) coding for high error environments.

4.1.6.1 Use of FEC and BERT in TACO2 transmissions. TACO2 transmissions and SIDS devices performing TACO2 transmissions shall comply with the requirements in Sections 4 and 5 of this document. FEC codes are defined in 5.4.2 and appendix C. These codes are required for compliance to the extent specified in 5.4.2.2. These codes are known as FEC-I and FEC-II. Section 5.4.2.3 specifies the detailed requirements for Bit Error Ratio Testing (BERT).

4.1.6.2 Placement of FEC within network protocol stack. The placement of FEC within the transmissions system may be approached in several ways. In most cases of practical importance, the FEC is situated within the OSI Data Link Layer, and as such will be below the level of the Network Layer (which in TACO2 is IP) and above the Physical Layer. Placements other than as part of the Data Link Layer are possible; these possibilities are not treated in this document.

4.1.6.3 Placement of FEC within data link layer. Three possible placements of FEC within the Data Link Layer are illustrated on figure 2. Each placement shown is applicable to NITFS transmissions. The placement shown on figure 2, known as packet coding, allows implementation in software and shall be the placement used for the FEC-I and FEC-II codes.

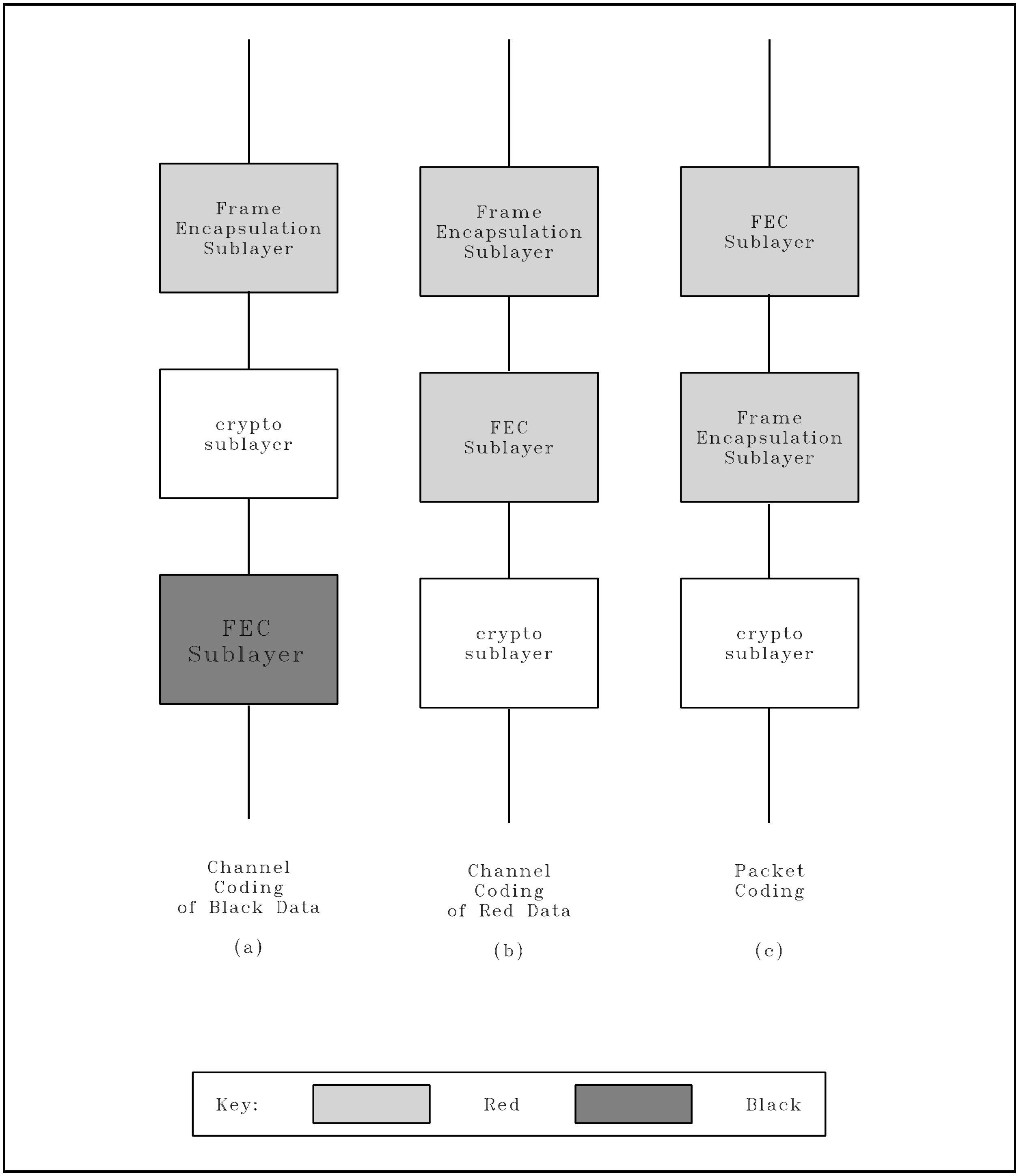

FIGURE 2. Possible positioning of the FEC within the data link layer.

4.1.7 HDLC and SLIP. HDLC and SLIP shall provide link-layer packet encapsulation for synchronous and asynchronous links respectively. TACO2 uses only the framing, bit-stuffing, and Cyclic Redundancy Check (CRC) components of HDLC. SLIP is a simple mechanism that provides encapsulation and byte-stuffing for use across asynchronous links.

4.1.8 Physical layer. The physical layer used with TACO2 depends on the cryptographic device and communications circuit used. The requirements for specific circuits are published as DISA/JIEO Technical Interface Specifications. Signal voltage sense shall be as specified in MIL-STD-188-114A.

4.2 Notation.

4.2.1 Hexadecimal representation. Throughout this document, a sequence of digits preceded by 0x (digit zero) is taken to be a hexadecimal integer. The hexadecimal digits include A through F, which represent the decimal values 10 through 15.

4.2.2 Data transmission order. The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets shall be the normal order in which they are read in English. On figure 3, the octets are transmitted in the order in which they are numbered.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

1	2	3	4
5	6	7	8
9	10	11	12

FIGURE 3. Transmission order of bytes.

Whenever an octet represents a numeric quantity, the left most bit in the diagram is the high order or most significant bit (MSB). The bit labeled 0 is the MSB, and the bit labeled 7 is the least significant bit (LSB). For example, figure 4 represents the value 106 (decimal), which is 0x6A (hexadecimal).

MSB							LSB
0	1	2	3	4	5	6	7
0	1	1	0	1	0	1	0

FIGURE 4. Significance of bits.

Similarly, whenever a multi-octet field represents a numeric quantity, the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet shall be transmitted first. Bit transmission order shall be as specified by the link layer (normally LSB first).

5. DETAILED REQUIREMENTS

TACO2 comprises the following protocol layers: the TACO2 NRTS, NETBLT, IP, Header Abbreviation, FEC, and either HDLC or SLIP. Detailed requirements for those protocol layers are provided in this section.

5.1 NITFS reliable transfer server for TACO2 (TACO2 NRTS). The TACO2 NRTS shall provide a (locally-defined) sending and receiving interface to the NITFS user process. It shall transfer (and if necessary act upon) the message and the metamessage to its peer TACO2 NRTS via NETBLT, and it may pass a destination IP address (at the message source) and exchange a set of parameters with NETBLT. The TACO2 NRTS described here assumes an active sender and a passive receiver ("push" operation); as of the effectivity date (Effectivity 4) the TACO2 NRTS also shall support an active receiver and passive sender ("pull" operation).

5.1.1 Metamessage definition. The metamessage shall be used to communicate information about the NITFS message between the source and the destination system (and intermediate systems, if any). It shall be transferred along with the NITFS message itself by TACO2, in a manner that allows the receiver to make an unambiguous connection between message and metamessage. In duplex communications, a return metamessage shall be transferred from receiver to sender, to allow possible negotiation or confirmation of some component values as specified in 5.1.1.2. The notation for the information about the message is designed to allow for additional attributes as the need arises, but with a minimum of additional complexity.

5.1.1.1 Description. The metamessage shall be a null-terminated string of 8-bit American Standard Code for Information Interchange (ASCII) letters, numbers, and some special characters (MIL-STD-2500 for the definition of the acceptable set of ASCII). The maximum length of the metamessage is unbounded (although implementations may have a limit on the length they can accept; this limit shall be at least 255 characters, and characters beyond the limit may be discarded), and the end of the metamessage shall be denoted by the occurrence of a null byte (hexadecimal 0x00). The individual components of the metamessage shall have the form:

<component-name> = <component-value>

where:

a. <component-name> shall be chosen from the list of components (see 5.1.1.2)

b. "=" is the literal ASCII character

c. <component-value> is a sequence of ASCII characters, not including the space or comma characters, providing the value for that component (numbers shall be given in ASCII decimal representation).

Components shall be separated by spaces or commas. The maximum length of the name components (message name, file name, sender name, and recipient name) is unbounded, although implementations may have a limit on the length they can accept; this limit shall be at least 32 characters for each such component. The only exception to the component form given above is the mandatory first component, Version, which shall be three bytes with hexadecimal value 0x5E0101. The number of occurrences of components that may occur multiple times is unbounded, but may be limited by the acceptable length of the metamessage. Components not specifically permitted to occur multiple times in a single metamessage shall occur no more than once in a metamessage.

5.1.1.2 Components. The following is a list of the metamessage components that may be supplied, and shall be recognized, by TACO2. Each component description includes an indication of whether it is optional or mandatory, its default value if it is absent, its form, and its meaning. Except for the mandatory initial component Version, the sequence of components in the metamessage is immaterial. Table I summarizes the metamessage components.

TABLE I. Metamessage components.

NAME	**FORMAT**	**PRESENCE**	**DEFAULT**
Version	0x5E0101	Mandatory	
Message Name	MNAME = < message-name >	Mandatory	None
File Name	FNAME = < file-name >	Optional	None
Message Type	TYP = < message-type >	Optional	NTF
Message Length	LEN = < message-length >	Optional	None
Start Point	STRT = < start-point >	Optional	0
Sender Name	FROM = < name >	Optional	None
Recipient Name	TO = < name >	Optional	None
Validity	NVAL = < first-octet > : < last-octet >	Optional	All valid
Criticality	CRIT = < first-octet > : < last-octet >	Optional	All equal
Geolocation	GEOG = < ddmmss > < N | S > < dddmmss > < E | W >	Optional	None

5.1.1.2.1 Version. (Mandatory). This shall be the first three bytes of the metamessage, and shall be hexadecimal 0x5E0101. The particular representation was chosen for compatibility with possible future alternative Abstract Syntax Notation One (ASN.1) representations of the metamessage.

5.1.1.2.2 Message name. (Mandatory, must occur within the first 255 characters of the metamessage). No default. MNAME = <message-name>, where <message-name> shall be a unique identifier for the NITFS message. (May be used for accountability tracking in some systems.) The <message-name> shall be unique for communication between two systems, as it will be used to identify (and possibly combine) multiple copies of the same message and to resume interrupted transfers (see 5.1.1.2.6). The originator may generate <message-name> with any method that meets the uniqueness requirement; file modification date-time may be included to help ensure uniqueness.

5.1.1.2.3 File name. (Optional). No default. FNAME = <file-name>, where <file-name> shall be a name under which the message may be stored in the destination system (it might be the name under which the originator stored it). The destination is under no obligation to use this name. If the receiver includes this component in a metamessage returned to the sender, <file-name> shall be the name under which the message will be stored in the receiving system.

5.1.1.2.4 Message type. (Optional). Default is NTF, signifying a NITF message. TYP = <message-type>, where <message-type> shall be the kind of message being transferred. NOTE: Other message types, such as text and spreadsheets, are To Be Resolved (TBR).

5.1.1.2.5 Message length. (Optional). No default. LEN = <message-length>, where <message-length> shall be a decimal number, represented in ASCII, describing the number of bytes in the NITFS message.

5.1.1.2.6 Start point. (Optional). Default = 0. STRT = <start-point>, where <start-point> shall be the decimal offset, represented in ASCII, of the first octet proposed to be transferred. (The octets in a file are numbered starting with 0). This may be used to resume interrupted transfers. If the receiver includes this component in a metamessage returned to the sender, <start-point> shall be the offset from the first octet in the file to be transferred, which shall be less than or equal to that proposed by the sender. In this case, the sender shall use the returned value as the actual start point of the transfer.

5.1.1.2.7 Sender name. (Optional). No default. FROM = <name>, where <name> shall be a meaningful ASCII name of the sender.

5.1.1.2.8 Recipient name. (Optional). No default. TO = <name>, where <name> shall be a meaningful ASCII name of a recipient. This component may occur multiple times in a single metamessage.

5.1.1.2.9 Validity. (Optional). Default is to assume that the entire message is valid. NVAL = <first-octet>:<last-octet>, where <first-octet> and <last-octet> shall be the inclusive boundaries of a possibly invalid portion of the message. The first octet of a message is numbered 0. This component may occur multiple times in a single metamessage. The receiver may use this information to assist in message interpretation.

5.1.1.2.10 Criticality. (Optional). Default is to assume that the entire message is equally critical. CRIT = < first-octet > : < last-octet > , where < first-octet > and < last-octet > shall be the inclusive boundaries of a portion of the message which is critical to the interpretation of the overall message. The first octet of a message is numbered 0. This component may occur multiple times in a single metamessage. Special actions may be taken to increase the probability of correct reception of this portion of the message (see 5.2.8.1.1).

5.1.1.2.11 Geolocation. (Optional). No default. GEOG = < ddmmss > < N | S > < dddmmss > < E | W > , with coordinates in the Geographic Coordinate System. The coordinate shall refer to the center of the image in the message. The first element represents degrees, minutes, and seconds of latitude; the second is N (north) or S (south); the third is degrees, minutes, and seconds of longitude; and the last is E (east) or W (west).

5.1.1.2.12 Additional components. Other components, with the same form, are reserved for future versions of TACO2.

5.1.1.2.13 Unrecognized components. An unrecognized component (such as, one that does not appear on this list) may be discarded by the recipient, and shall not disrupt the protocol exchange.

5.1.2 TACO2 NRTS operation. This section describes the interaction between the TACO2 NRTS and NETBLT. The details of the interaction and interfaces between protocol layers are not specified herein; however, the complete TACO2 protocol stack shall interact with other TACO2 protocol stack implementations in a manner consistent with this description. The descriptions in this section are supplemented with time sequence diagrams, which illustrate the relationship between operations across the TACO2 NRTS to NETBLT interface. The vertical lines represent the interfaces, and time increases down the diagram.

5.1.2.1 Summary of operation. When a message is to be transferred, the TACO2 NRTS requests that NETBLT open a connection to the intended receiver. Once the connection is opened successfully, the message is transferred between the TACO2 NRTS and NETBLT as a series of buffers at both the sending and receiving end. All buffers except the last one shall be the same size. When the transfer is complete, the NETBLT connection is closed. If the entire message is not transferred successfully, the sending TACO2 NRTS may attempt to open a new connection and restart the transfer (see 5.1.1.2.6). Status information not described here may be provided by NETBLT to the TACO2 NRTS, and by the TACO2 NRTS to the user process.

5.1.2.2 Connection establishment. The sequence of events by which the TACO2 NRTS establishes a connection via NETBLT is shown on figure 5.

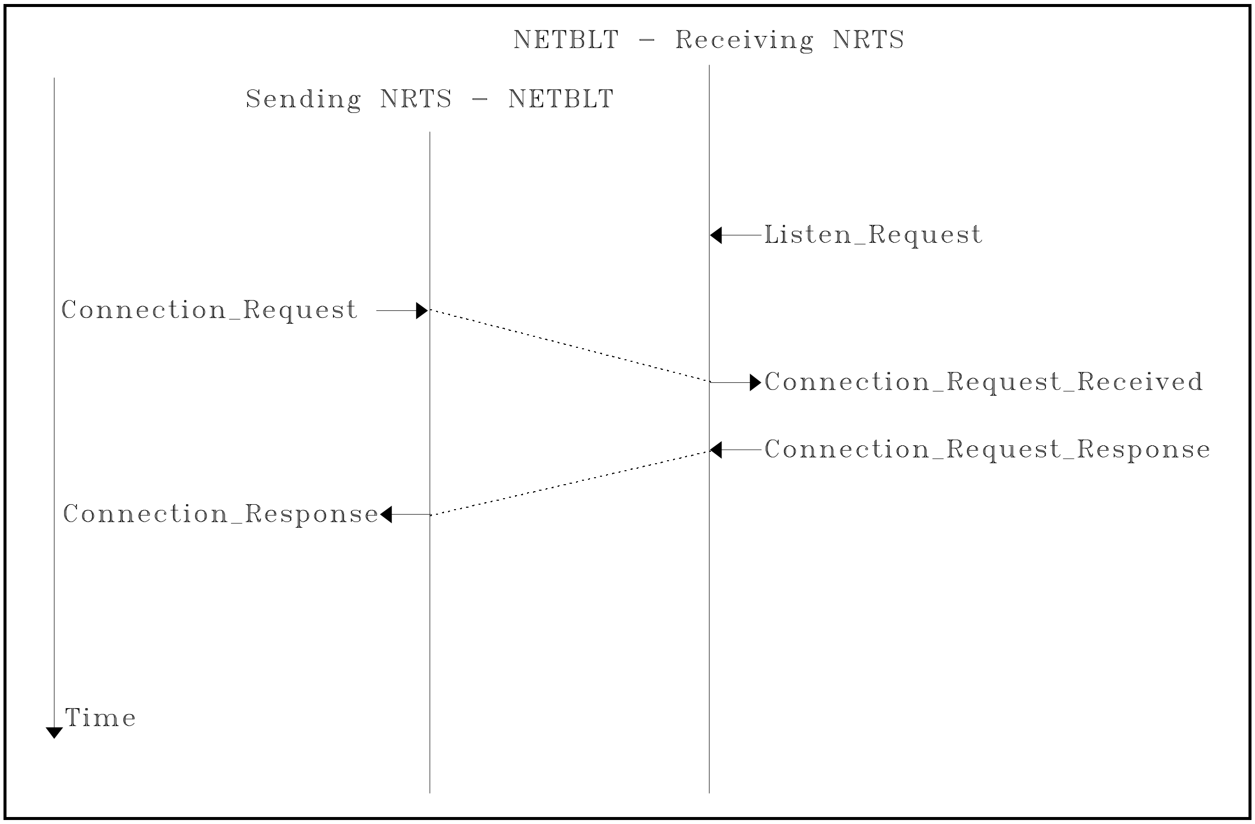

FIGURE 5. TACO2 NRTS connection establishment.

5.1.2.2.1 Sending end system.

5.1.2.2.1.1 Connection_Request. The TACO2 NRTS shall request that NETBLT open a connection to the intended receiver with a Connection_Request, which has the following parameters:

a. Foreign host. A 32-bit IP address for the destination of the NITFS message.

b. Foreign port. Shall be 1.

c. Checksum flag. Shall require data checksumming.

d. Send/Receive flag. Shall indicate send.

e. Metamessage. Shall be as specified in 5.1.1

f. Proposed maximum number of outstanding buffers.

g. Proposed buffer size.

5.1.2.2.1.2 Connection_Response. NETBLT shall respond to the Connection_Request from the TACO2 NRTS with Connection_Response, which has two parameters:

a. Success/failure flag.

b. Return metamessage, if successful; reason for failure, if unsuccessful.

5.1.2.2.2 Receiving end system.

5.1.2.2.2.1 Listen_Request. When the TACO2 NRTS is prepared to receive a message, it shall invoke NETBLT with a Listen_Request, which has the following parameters:

a. Local port. Shall be 1.

b. Maximum acceptable number of outstanding buffers.

c. Maximum acceptable buffer size.

5.1.2.2.2.2 Connection_Request_Received. When a connection is opened by a foreign host, NETBLT shall respond to the Listen_Request of the receiving-side TACO2 NRTS with Connection_Request_Received, which has the following parameters:

a. Foreign host. The 32-bit IP address of the source of the NITFS message.

b. Send/receive flag. Should indicate send (by the other end).

c. Metamessage. Shall be the metamessage provided by the sending TACO2 NRTS.

5.1.2.2.2.3 Connection_Request_Response. The TACO2 NRTS shall respond to NETBLT with Connection_Request_Response, which has two parameters:

a. Accept/reject flag.

b. Return metamessage, as specified in 5.1.1, if successful; reason for rejection, if unsuccessful.

5.1.2.3 Data transfer. The sequence of actions by which the TACO2 NRTS transfers a message via NETBLT is shown on figure 6. This sequence shall be repeated until all data in the message have been transferred, unless the transfer is terminated prematurely.

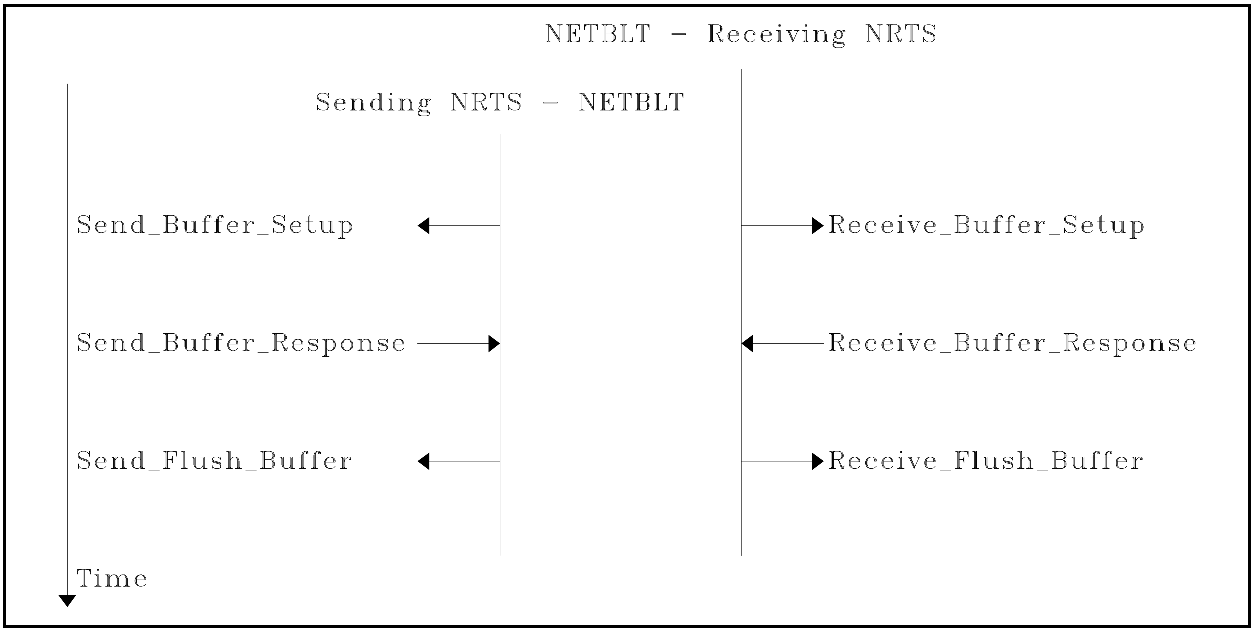

FIGURE 6. TACO2 NRTS data transfer.

5.1.2.3.1 Sending end system.

5.1.2.3.1.1 Send_Buffer_Setup. The sending NETBLT shall request the next buffer full of data from the TACO2 NRTS with Send_Buffer_Setup, which has two parameters:

a. Buffer number, where buffers are numbered consecutively starting with 0.

b. Requested size of the buffer.

5.1.2.3.1.2 Send_Buffer_Response. The TACO2 NRTS shall respond to NETBLT with Send_Buffer_Response, which has the following parameters:

a. Identification of the buffer with data to be sent. The data in that buffer shall be consecutive bytes of the message to be transferred, starting with byte number ((buffer number-1)*(requested size of the buffer)), where bytes are numbered starting with 0.

b. Actual size of the buffer.

c. Last buffer flag. Set if this is the last buffer of the message.

5.1.2.3.1.3 Send_Flush_Buffer. When NETBLT has finished with the buffer, it shall invoke Send_Flush_Buffer, which has one parameter: identification of the buffer.

5.1.2.3.2 Receiving end system.

5.1.2.3.2.1 Receive_Buffer_Setup. The receiving NETBLT shall request an empty buffer from the TACO2 NRTS with Receive_Buffer_Setup, which has two parameters:

a. Buffer number.

b. Requested size of the buffer.

5.1.2.3.2.2 Receive_Buffer_Response. The TACO2 NRTS shall respond to NETBLT with Receive_Buffer_Response, which has one parameter: identification of the buffer.

5.1.2.3.2.3 Receive_Flush_Buffer. When NETBLT has filled the buffer, it shall invoke the TACO2 NRTS with Receive_Flush_Buffer, which has the following parameters:

a. Identification of the buffer. The data in that buffer shall be consecutive bytes of the message to be transferred, starting with byte number ((buffer number-1)*(requested size of the buffer)), where bytes are numbered starting with 0. Received buffers may be returned out of order by NETBLT; the correct placement of data in the received message shall be established by the TACO2 NRTS, by correctly associating the buffer identification and the buffer number.

b. Actual size of the buffer.

c. Last buffer flag. Set if this is the last buffer of the message.

5.1.2.4 Connection termination. A TACO2 message transfer may be terminated by NETBLT or by the TACO2 NRTS. The possible terminations are shown on figure 7.

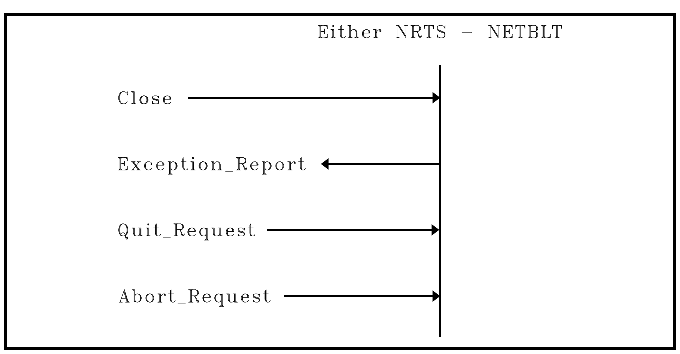

FIGURE 7. TACO2 NRTS connection termination.

5.1.2.4.1 NETBLT-invoked termination.

5.1.2.4.1.1 Close. After all buffers comprising the message have been transferred, NETBLT shall notify both the sending and receiving NRTS with a Close, which has one parameter: List of possibly invalid bytes (this parameter is only meaningful in receive simplex operation; in any other operation, it shall be empty).

5.1.2.4.1.2 Exception_Report. If NETBLT is unable to complete the transfer, it shall send an Exception_Report to the TACO2 NRTS. Exception_Report has one parameter: Reason for the report.

5.1.2.4.2 TACO2 NRTS-invoked termination.

5.1.2.4.2.1 Quit-request. The TACO2 NRTS shall invoke Quit_Request to cause an orderly but premature termination of a TACO2 message transfer. Quit_Request has one parameter: Reason for request.

5.1.2.4.2.2 Abort-request. The TACO2 NRTS shall invoke Abort-Request to cause an immediate termination of a TACO2 message transfer. Abort_Request has one parameter: Reason for request.

5.2 Transport layer - NETBLT. The bulk data transfer protocol NETBLT forms the transport layer of the TACO2 protocol suite. NETBLT in TACO2 works by opening a connection between a sender and a receiver (at the sender's request), transferring a message in one or more buffers, and closing the connection. Each buffer is transferred as a sequence of packets; the interaction between sender and receiver is primarily on a per-buffer basis. An overview of NETBLT is provided in 5.2.1; further explanation and detailed requirements are found in the following sections. The material here assumes the existence of a full-duplex connection between sender and receiver, where packets can be transferred in both directions concurrently. Changes for half-duplex and simplex cases are provided in 5.2.7. Specific packet types are identified in the following sections by upper-case names (DATA packets), in contrast with packet functions (keepalive packets), which are accomplished by more than one packet type.

5.2.1 NETBLT full-duplex operations. NETBLT opens a connection between two "clients" (the "sender" and the "receiver"), transfers the data in a series of large data aggregates called "buffers," and then closes the connection. (In TACO2, those clients are the TACO2 NRTS layer). NETBLT transfers each buffer as a sequence of packets.

5.2.1.1 Single-buffer operation. In its simplest form, a NETBLT transfer works as follows: the sending client loads a buffer of data and passes it to the NETBLT layer to be transferred. NETBLT breaks the buffer up into packets and sends these packets across the network via IP datagrams. The receiving NETBLT loads these packets into a matching buffer. When the last packet in the buffer should have arrived at the receiver, the receiving NETBLT checks whether all packets in that buffer have been received correctly. If some packets have not been received correctly, the receiving NETBLT requests that they be resent. When the buffer has been received completely, the receiving NETBLT passes it to the receiving client. When a new buffer is ready to receive more data, the receiving NETBLT notifies the sender that the new buffer is ready, and the sender prepares and sends the next buffer in the same manner. This continues until all the data has been sent; at that time the sender notifies the receiver that the transmission has been completed. The two sides then close the connection.

5.2.1.2 Multiple-buffer operation. As described in 5.2.1.1, the NETBLT protocol is "lock-step." Action halts after a buffer is transmitted, and begins again after confirmation is received from the receiver of data. NETBLT provides for multiple buffering, so that the sending NETBLT can transmit new buffers while earlier buffers are waiting for confirmation from the receiving NETBLT.

5.2.2 Buffers and packets.

5.2.2.1 Buffers. The data to be transmitted shall be broken into buffers by the sending client. Each buffer shall be the same size, except for the last buffer. During connection setup, the sending and receiving NETBLTs shall negotiate the buffer size. Buffer sizes shall be in bytes only; data shall be placed in buffers on byte boundaries.

5.2.2.2 Packets. Buffers are broken down by NETBLT into sequences of DATA packets. DATA packet size shall be negotiated between the sending and receiving NETBLTs during connection setup. All DATA packets shall be the same size. The last packet of every buffer is not a DATA packet but rather an LDATA packet. DATA and LDATA packets are identical in format except for the packet type.

5.2.3 Flow control. NETBLT uses two strategies for flow control, one at the client level and one internal.

5.2.3.1 Client level flow control. The sending and receiving NETBLTs transmit data in buffers; therefore, client flow control is by buffer. Before a buffer can be transmitted, NETBLT confirms that both clients have set up matching buffers, that one is ready to send data, and that the other is ready to receive data. Either client can control data flow by not providing a new buffer. Clients cannot stop a buffer transfer once it is in progress, except by aborting the entire transfer.

5.2.3.2 Internal flow control. The internal flow control mechanism for NETBLT is rate control. The transmission rate is negotiated by the sending and receiving NETBLTs during connection setup and after each buffer transmission. The sender uses timers to maintain the negotiated rate, by controlling the time to transmit groups of packets. The sender transmits a burst of packets over the negotiated time interval, and sends another burst in the next interval, therefore, NETBLT's rate control has two parts, a burst size and a burst interval, with (burst interval)/(burst size) equal to the average transmission time per packet.

5.2.3.3 Flow control parameter negotiation. All NETBLT flow control parameters (packet size, buffer size, number of buffers outstanding, burst size, and burst interval) shall be negotiated during connection setup. The negotiation process is the same for all parameters. The client initiating the connection (the sender) shall send a value for each parameter in its OPEN packet. The other client (the receiver) shall compare these values with the highest-performance values it can support. The receiver can modify any of the parameters, but only by making them more restrictive, such as, smaller packet size, smaller buffer size, fewer buffers, smaller burst size, and larger burst interval. The (possibly modified) parameters shall be sent back to the sender in the RESPONSE packet.

5.2.3.4 Flow control parameter renegotiation. The burst size and burst interval also may be re-negotiated after each buffer transmission to adjust the transfer rate according to the performance observed from transferring the previous buffer. The receiving end shall send burst size and burst interval values in its OK messages (described in 5.2.5.2.1). The sender shall compare these values with the values it can support. It then may modify either of the parameters, but only by making them more restrictive. The modified parameters shall then be communicated to the receiver in DATA, LDATA, or NULL-ACK packets.

5.2.3.5 Client-controlled flow. (Effectivity 3). A burst interval value of zero shall have a special meaning: internal flow control shall be turned off, so that only client level flow control shall be in effect. In this case, the sending NETBLT shall transmit packets without regard for the rate control mechanism. When using client-controlled flow, the receiver shall use an alternative method for data timer estimation (see 5.2.5.2.4.3).

5.2.4 Checksumming. NETBLT shall use checksums to validate the contents of packets and packet headers unless packet integrity is assured by the data link layer. The checksum value shall be the bitwise negation of the ones-complement sum of the 16-bit words being checksummed. On twos-complement machines, the ones-complement sum can be computed by means of an "end around carry"; that is, any overflows from the most significant bit are added into the least significant bits. See figure 8 for an example. If the quantity to be checksummed has an odd number of bytes, it shall be padded with a final null byte (binary 0's) to make the number of bytes even for the purpose of checksum calculation. The extra byte shall not be transmitted as part of the packet, but its existence shall be assumed at the receiving end for checksum verification.

	Word 1	0001	(Note: all numbers are in hexadecimal)
	Word 2	F203	
	Word 3	F4F5	
	Word 4	F6F7	
Sum showing carry:		2DDF0	
Sum without carry:		DDF0	
	Carry:	2	
Ones-complement sum:		DDF2	
Complement for checksum:		220D	

FIGURE 8. Example of checksumming.

5.2.5 NETBLT protocol operation. Each NETBLT transfer shall have three stages: connection setup, data transfer, and connection close. The stages are described in detail below, along with methods for ensuring that each stage completes reliably. State diagrams are provided at the end of the description for each stage of the transfer. Each transition in the diagrams is labeled with the event that causes the transition, and optionally, in parentheses, actions that occur at the time of the transition.

5.2.5.1 Connection setup. A NETBLT connection shall be set up by an exchange of two packets between the sending NETBLT and the receiving NETBLT. The sending end shall send an OPEN packet; the receiving end shall acknowledge the OPEN packet in one of two ways: it shall either send a REFUSED packet, indicating that the connection cannot be completed for some reason, or it shall complete the connection setup by sending a RESPONSE packet. After a successful connection setup, the transfer can begin. Figure 9 illustrates the opening of a connection by a sender, and figure 10 shows the same process for a receiver.

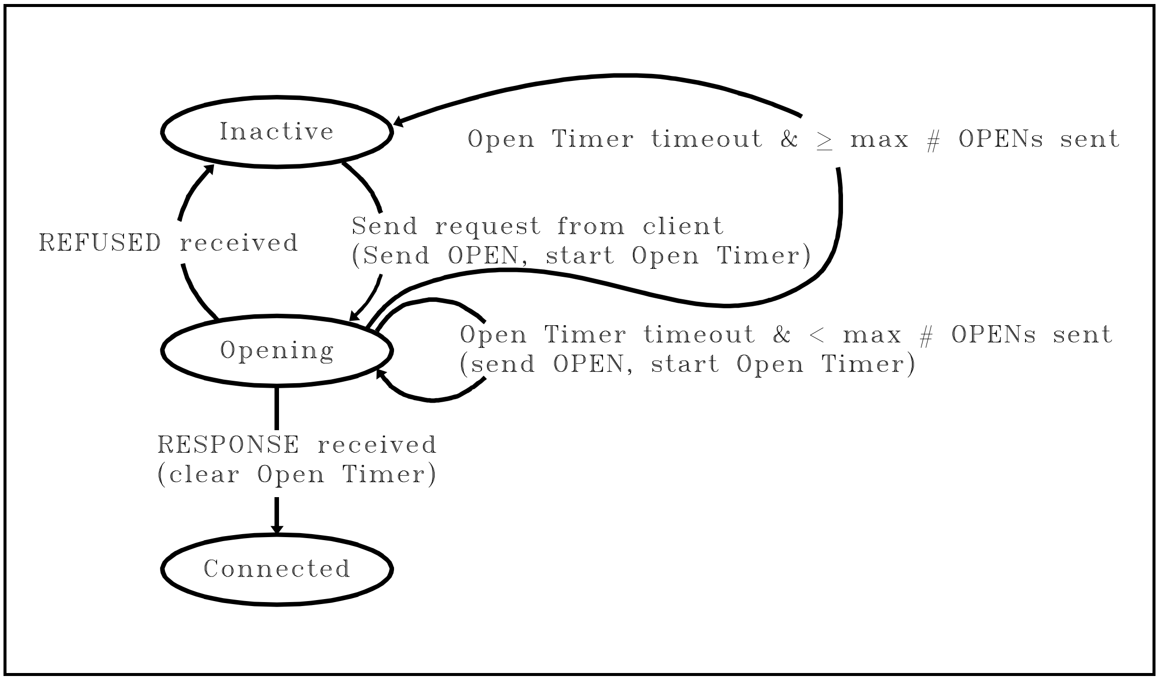

FIGURE 9. Sender open state diagram.

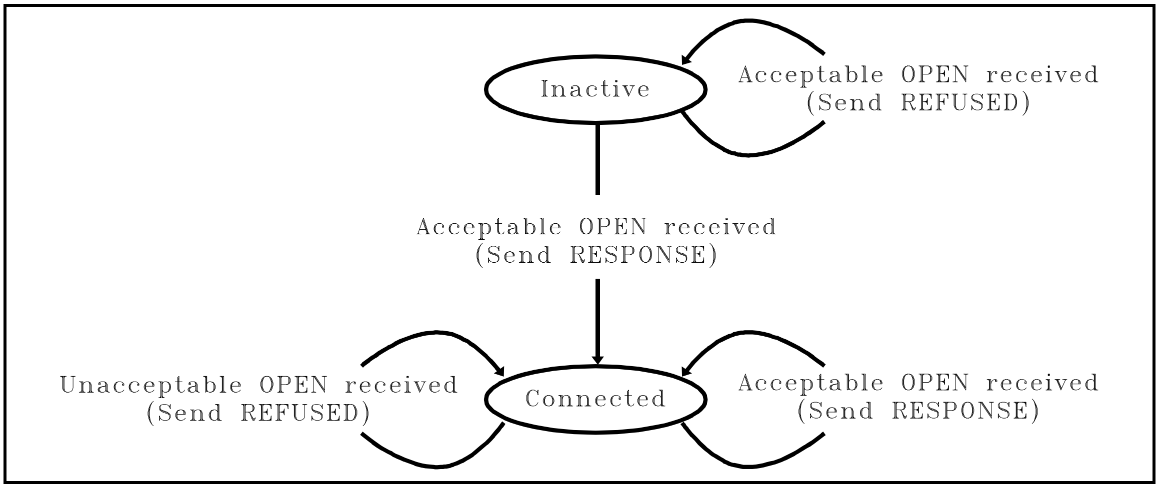

FIGURE 10. Receiver open state diagram.

5.2.5.1.2 Death timeout. Each side of the connection shall transmit its death-timeout value in seconds in the OPEN or the RESPONSE packet. The death-timeout value shall be used to determine the frequency with which to send keepalive packets during idle periods of an opened connection (death timers and keepalive packets are described in 5.2.5.2.5 and 5.2.5.2.6).

5.2.5.1.3 Port number. The sending NETBLT shall specify a passive client through a client-specific "well-known" (see 5.2.9.1.2) 16-bit logical port number on which the receiving end listens. The sending client shall identify itself through a 32-bit Internet address and a locally unique 16-bit port number.

5.2.5.1.4 Client string. An unstructured, variable-length client message field is provided in OPEN and RESPONSE packets for any client-specific information that may be required. In TACO2, this field shall be used to transfer the metamessage provided by the TACO2 NRTS. In addition, a "reason for refusal" field is provided in REFUSED packets.

5.2.5.1.5 OPEN timer. Recovery for lost OPEN and RESPONSE packets shall be provided using timers. The sending end shall set a timer when it sends an OPEN packet. When the timer expires, another OPEN packet shall be sent, until some predetermined maximum number (at least five) of OPEN packets have been sent. The timer shall be cleared upon receipt of a RESPONSE or REFUSED packet.

5.2.5.1.6 Connection ID. To prevent duplication of OPEN and RESPONSE packets, the OPEN packet contains a 32-bit connection unique identifier (UID) that must be returned in the RESPONSE packet. This UID prevents the initiator from confusing the response to the current request with the response to an earlier connection request (there can only be one connection open between any pair of logical ports). Any OPEN or RESPONSE packet with a port pair matching that of an open connection shall have its UID checked. If the UID of the packet matches the UID of the connection, then the packet type shall be checked. If it is a RESPONSE packet, it shall be treated as a duplicate and ignored. If it is an OPEN packet, the passive NETBLT shall send another RESPONSE (on the assumption that a previous RESPONSE packet was sent and lost, causing the initiating NETBLT to retransmit its OPEN packet). A non-matching UID shall be treated as an attempt to open a second connection between the two ports and shall be rejected by sending a REFUSED message.

5.2.5.2 Data transfer.

5.2.5.2.1 Single buffer transfer. The simplest full-duplex mode of data transfer shall proceed as follows. The sending client shall set up a buffer full of data. The receiving NETBLT shall send a GO message inside a CONTROL packet to the sender, signifying that it too has set up a buffer and is ready to receive data. Once the GO message is received, the sender shall transmit the buffer as a series of DATA packets followed by an LDATA packet. When the last packet in the buffer should have been received (as determined by a timer, see 5.2.5.2.4.2), if any packets of that buffer have not been received, the receiver shall send a RESEND message inside a CONTROL packet containing a list of packets that were not received. The sender shall resend these packets. This process shall continue until there are no missing packets. At that time the receiver shall send an OK message inside a CONTROL packet, shall set up another buffer to receive data, and shall send another GO message. The sender, having received the OK message, shall set up another buffer, wait for the GO message, and repeat the process.

5.2.5.2.2 Multiple buffer transfer. A more efficient full-duplex transfer mode uses multiple buffering, in which the sender and receiver allocate and transfer buffers in a manner that allows error recovery or successful transmission confirmation of previous buffers to be concurrent with transmission of the current buffer. During the connection setup phase, one of the negotiated parameters is the number of concurrent buffers permitted during the transfer. If more than one buffer is available, transfer of the next buffer shall start right after the current buffer finishes, and the receiver is ready to receive the buffer. The receiver shall signal that it is ready for the next buffer by sending a GO message. This is illustrated in the following example. Assume the sender has received two buffers, A and B, in a multiple-buffer transfer, with A preceding B. When A has been transferred and the sending NETBLT is waiting for either an OK or a RESEND message for it, the sending NETBLT can start sending B immediately,. If the receiver of data sends an OK for A, all is well; if it receives a RESEND, the missing packets specified in the RESEND message shall be retransmitted. In the multiple-buffer transfer mode, all packets to be sent shall be ordered by buffer number (lowest number first). Since buffer numbers increase monotonically, packets from an earlier buffer shall always precede packets from a later buffer.

5.2.5.2.3 Recovering from lost control messages.

5.2.5.2.3.1 Control packet. NETBLT shall use a single long-lived control packet; the packet shall be treated like a First-In-First-Out queue, with new control messages added on at the end and acknowledged control messages removed from the front. The implementation shall place control messages in the control packet and shall transmit the entire control packet, consisting of any unacknowledged control messages plus new messages just added. The entire control packet shall also be transmitted whenever the control timer expires. Since control packet transmissions are fairly frequent, unacknowledged messages may be transmitted several times before they finally are acknowledged. The receiver may send zero or more control messages (OK, GO, or RESEND) within a single CONTROL packet.

5.2.5.2.3.2 Control timer. Whenever the receiver sends a control packet, it shall start a control timer. When the control timer expires, the receiving NETBLT shall resend the control packet and reset the timer. The receiving NETBLT shall continue to resend control packets in response to control timer expiration until either the control timer is cleared or the receiving NETBLT's death timer (described in 5.2.5.2.5) expires (at which time it shall shut down the connection). The appropriate initial control timer value is constrained by the characteristics of the link. Subsequent control packets shall have their timer values based on the network round-trip transit time (the time between sending the control packet and receiving the acknowledgment of all messages in the control packet) plus a variance factor. The timer value shall be updated continually, based on a smoothed average of collected round-trip transit times. The control timer shall be set to the keepalive value when a packet is received from the sender with high-acknowledged-sequence-number equal to the highest sequence number in the control packet most recently sent (see 5.2.5.2.3.3). The current value of the control timer is provided to the sender in each OK message.

NOTE: The exact algorithm for control timer calculation is not a required part of this standard. The recommended algorithm is as follows:

a. Initially, the round trip time is set to the keepalive value and the deviation is set to zero.

b. Thereafter, new smoothed round trip time = (1-a) * old smoothed round trip time + a * latest round trip measurement

c. New deviation = (1-b) * old deviation + b * |latest round trip measurement - old smoothed round trip time|

where a = 1/8 and b = 1/4, allowing computations to be done with add and shift operations. The control timer is set equal to the new smoothed round trip time plus twice the new deviation, or to the keepalive value, whichever is less, if the control packet is not empty. If the control packet is empty, the control timer is set to the keepalive value.

5.2.5.2.3.3 Control message sequence number. Each control message shall include a sequence number, which starts at one and increases by one for each control message sent. The sending NETBLT shall check the sequence number of every incoming control message against all other sequence numbers it has received. It shall store the highest sequence number below which all other received sequence numbers are consecutive (in following paragraphs this is called the high-acknowledged-sequence-number) and shall return this number in every packet flowing back to the receiver. The receiver shall remove control messages with sequence numbers less than or equal to the high-acknowledged-sequence-number from the control packet. The receiver shall set its control timer to the keepalive value (see 5.2.5.2.6) when it receives a packet from the sender with a high-acknowledged-sequence-number equal to the highest sequence number in the control packet just sent.

5.2.5.2.3.4 Response to control messages. The sending NETBLT, upon receiving a CONTROL packet, shall either set up a new buffer (upon receipt of an OK message for a previous buffer), mark data for resending (upon receipt of a RESEND message), or prepare a buffer for sending (upon receipt of a GO message). If the sending NETBLT is not in a position to send data, it shall send a NULL-ACK packet, which contains its high-acknowledged-sequence-number (this permits the receiving NETBLT to acknowledge any outstanding control messages), and wait until it can send more data.

5.2.5.2.4 Recovering from lost DATA and LDATA packets. NETBLT solves the problem of DATA and LDATA packet loss by using a data timer for each buffer at the receiving end. The simplest data timer model has a data timer set when a buffer is ready to be received; if the data timer expires, the receiving NETBLT shall send a RESEND message requesting all missing DATA/LDATA packets in the buffer. When all packets have been received, the timer shall be cleared. Data timer values shall be based on the amount of time taken to transfer a buffer (as determined by the number of DATA packet bursts in the buffer times the burst interval) plus a variance factor.

5.2.5.2.4.1 Send buffer state sequence. The state sequence for a sending buffer shall be as follows: when a GO message for the buffer is received, the buffer is created, filled with data, and placed in a

SENDING state. When an OK for that buffer has been received, it goes into a SENT state and may be released. Figure 11 illustrates this sequence.

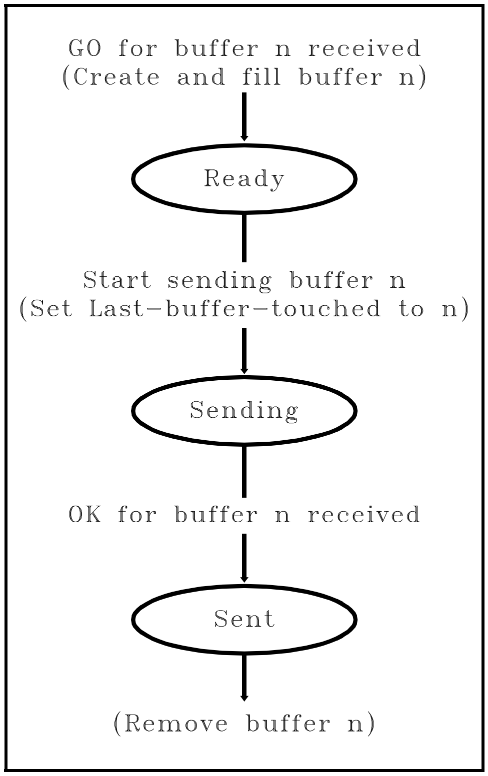

FIGURE 11. Sending buffer state diagram.

5.2.5.2.4.2 Receive buffer state sequence. The state sequence for a receiving buffer is a little more complicated. Assume existence of Buffer A. When a control message for Buffer A is sent, the buffer shall move into state ACK-WAIT (it is waiting for acknowledgement of the control message). As soon as the control message has been acknowledged, Buffer A shall move from the ACK-WAIT state into the ACKED state (it is now waiting for DATA packets to arrive). At this point, the control message shall be removed from the control packet. Buffer A shall stay in the ACKED state until a DATA, LDATA, or NULL-ACK packet arrives with its "Last Buffer Touched" number greater than or equal to Buffer A's number. At this time, Buffer A's data timer shall be set to the time expected for the remaining packets in the buffer to be received plus a variance, and Buffer A shall move to the RECEIVING state. (Note: This mechanism is different from, and simpler than, the "loose/tight" timer mechanism described in Request for Comment (RFC) 998). When all DATA packets for A have been received, it shall move from the RECEIVING state to the RECEIVED state and may be passed to the receiving client. Had any packets been missing, Buffer A's data timer would have expired; in that case, Buffer A shall move into the ACK-WAIT state after sending a RESEND message. The sending of a RESEND message shall cause the data timers of all buffers currently in the RECEIVING state to be recalculated, since the presence of resend packets will change the expected completion time for later buffers. The state progression would then move as in the above example. Figure 12 illustrates this sequence.

NOTE: The exact algorithm for data timer estimation is not a required part of this standard. The recommended algorithm is to compute the number of packets expected before the buffer is complete, multiply that by the time required to transmit a packet, and add a variance of 50 percent. The time required to transmit a packet is the burst interval divided by the burst size.

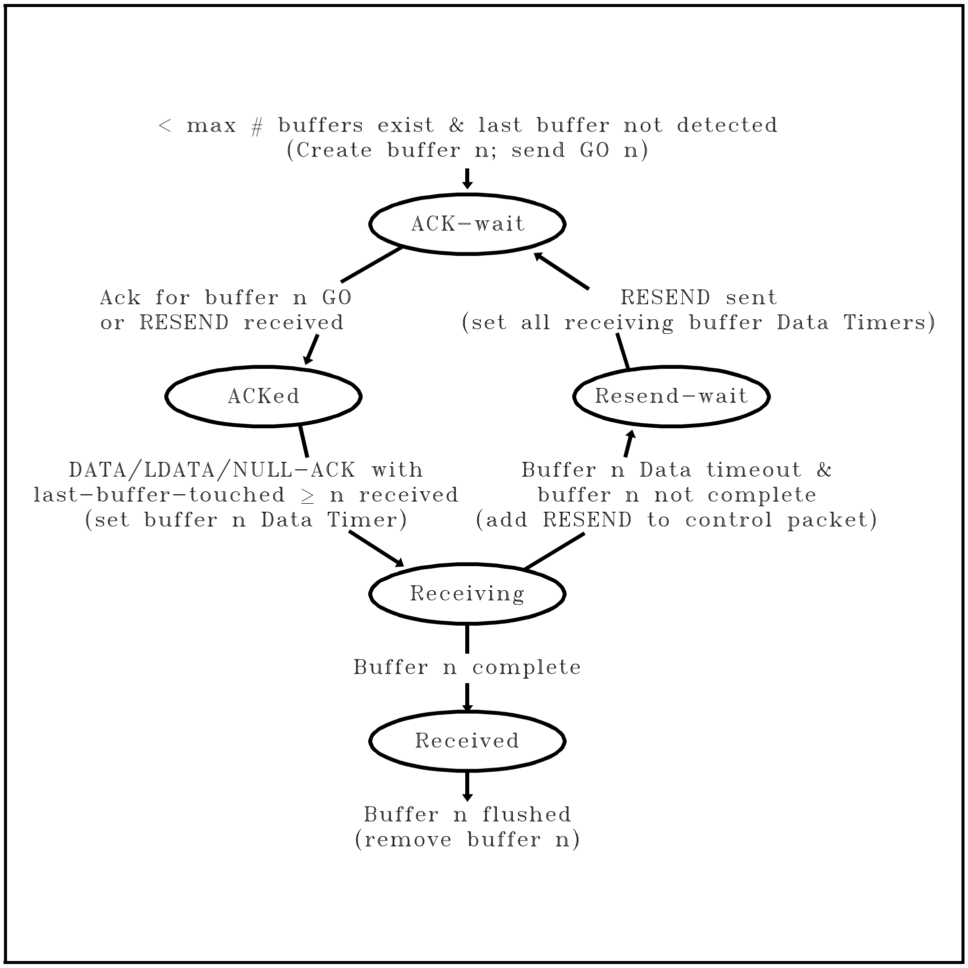

FIGURE 12. Receiving buffer state diagram.

5.2.5.2.4.3 Alternative method for data timer estimation. When the burst interval is set to zero, an alternative method for estimating the time required to send a packet shall be used for the purpose of data timer estimation. The alternative algorithm is to measure the elapsed time to receive a consecutively-numbered sequence of packets, and to divide that time by the number of packets in the sequence to calculate the time required for transmission of a single packet. This value may then be used in the algorithm recommended in 5.2.5.2.4.2. This value may be smoothed, in a manner consistent with that recommended for control timer calculation in 5.2.5.2.3.2. A suggested initial value used by the receiver for the time required to transmit a single packet is (sender's death timeout value * packet size) / (buffer size * maximum outstanding buffers * 4).

5.2.5.2.5 Death timers. At connection startup, each NETBLT shall send its death value to the other end in the OPEN or the RESPONSE packet. As soon as the connection is opened, each end shall set its death timer to its chosen value; this timer shall be reset every time a packet is received. When a NETBLT's death timer expires, it shall close the connection without sending any more packets. A recommended value for the death timer is in 5.2.9.10.

5.2.5.2.6 Keepalive packets. NETBLT shall include a keepalive function, which sends packets repeatedly at fixed intervals when a NETBLT has no other reason to send packets. The sender shall use NULL-ACKs as keepalive packets; the receiver shall use empty CONTROL packets. If the sending NETBLT is not ready to send upon receipt of a control packet, it shall send a single NULL-ACK packet to clear any outstanding control timers at the receiving end. Each end shall use the other end's death-timeout value to compute a frequency with which to send keepalive packets. The keepalive frequency shall be high enough that several keepalive packets can be lost before the other end's death timer expires; recommended values are the sender's death timer value divided by seven for the receiver, and the receiver's death timer value divided by eight for the sender. Keepalive intervals should be different to avoid repeated collisions in half-duplex operations.

5.2.5.3 Terminating the connection. The four conditions under which a connection shall be terminated are a successful transfer, a client quit or abort, a NETBLT abort, and a death timer timeout.

5.2.5.3.1 Successful transfer After a successful data transfer, NETBLT shall close the connection.

5.2.5.3.1.1 Receiver successful close. When the sender is transmitting the last buffer of data, it shall set a "last-buffer" flag on every DATA packet in the buffer. The receiver shall recognize that the transfer has completed successfully when all of the following are true: (1) it has received DATA packets with a "last-buffer" flag set, (2) all its control messages have been acknowledged, and (3) it has no outstanding buffers with missing packets. The DONE packet shall be transmitted when the receiver recognizes that the transfer has been completed successfully. At that point, the receiver shall close its half of the connection. Figure 13 illustrates this sequence.

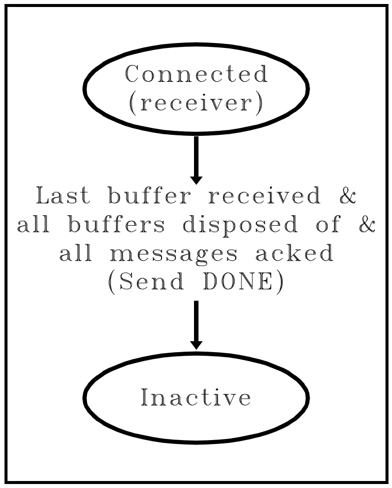

FIGURE 13. Receiver successful close state diagram.

5.2.5.3.1.2 Sender successful close. The sender shall recognize that the transfer has completed when the following are true: (1) it has transmitted DATA packets with a "last-buffer" flag set and (2) it has received OK messages for all its buffers. At that point, it shall "dally" for a predetermined period of time before closing its half of the connection. If the NULL-ACK packet acknowledging the receiver's last OK message was lost, the receiver has time to retransmit the OK message, receive a new NULL-ACK, and recognize a successful transfer. The dally timer value shall be based on the receiver's control timer value; it shall be long enough to allow the receiver's control timer to expire so that the OK message can be resent. The sender shall use the receiver's current control timer value to compute its dally timer value. A value of twice the receiver's control timer value is suitable for the dally timer. When the sender receives a DONE packet, it shall clear its dally timer and close its half of the connection. Figure 14 illustrates this sequence.

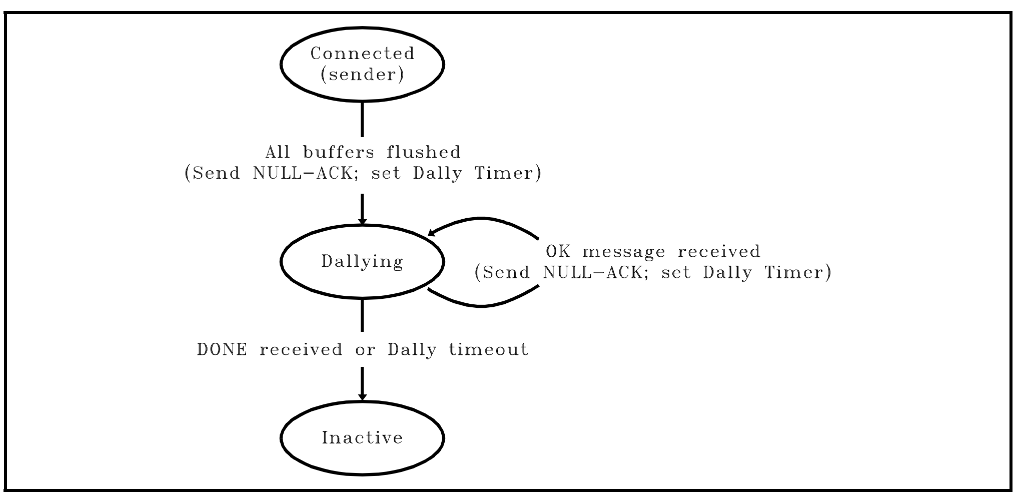

FIGURE 14. Sender successful close state diagram.

5.2.5.3.2 Client QUIT. During a NETBLT transfer, one client may send a QUIT packet to the other, to terminate the transfer prematurely. Since the QUIT occurs at a client level, the QUIT transmission shall occur only between buffer transmissions. The NETBLT receiving the QUIT packet shall take no action other than immediately notifying its client and transmitting a QUITACK packet. The QUIT sender shall time out and retransmit until a QUITACK has been received or its death timer expires. The sender of the QUITACK shall dally before quitting, so that it can respond to a retransmitted QUIT. Figure 15 illustrates this sequence.

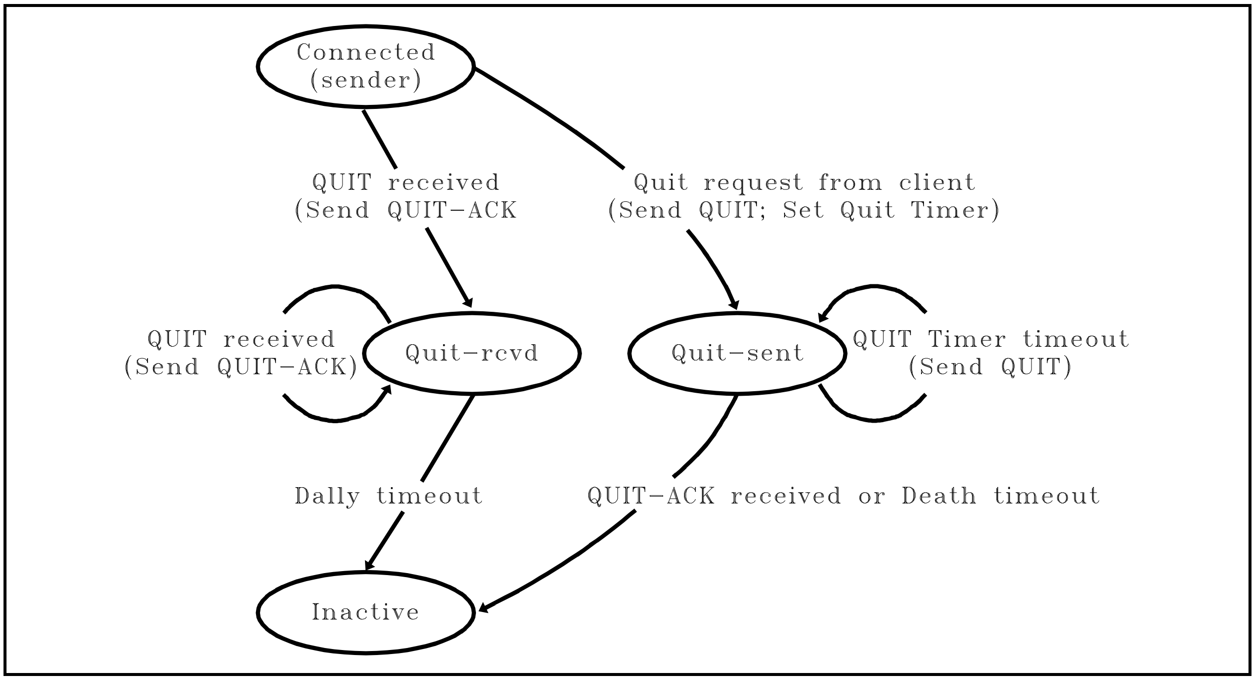

FIGURE 15. Quit state diagram.

5.2.5.3.3 NETBLT ABORT. An ABORT shall take place when an unrecoverable malfunction occurs. Since the ABORT originates in the NETBLT layer, it may be sent at any time. The ABORT implies that the NETBLT layer is malfunctioning, so no transmit reliability is expected, and the sender shall immediately close its connection. Figure 16 illustrates this sequence.

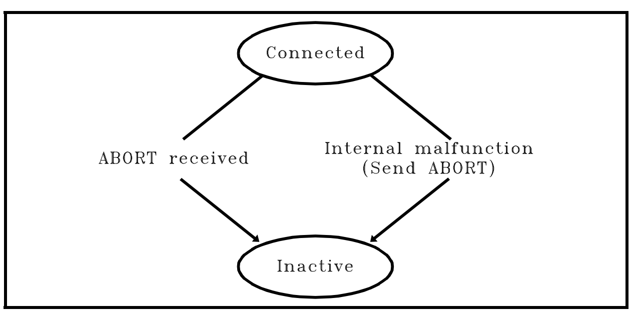

FIGURE 16. Abort state diagram.

5.2.5.3.4 Death timer timeout. When a NETBLT's death timer expires, it shall close the connection without sending further packets.

5.2.6 Protocol layering structure. NETBLT shall be implemented directly on top of the IP. It has been assigned an official protocol number of 30 (decimal), which is 0x1E (hexadecimal).

5.2.7 Packet formats. NETBLT packet formats used in TACO2 shall be in accordance with those given in this section. NETBLT packets are divided into three categories, all of which share a common 12-byte packet header.

a. There are three packet types that travel only from data sender to receiver; these contain the high-acknowledged-sequence-numbers which the receiver uses for control message transmission reliability. These are the NULL-ACK, DATA, and LDATA packets.

b. There is one packet type that travels only from receiver to sender. This is the CONTROL packet. Each CONTROL packet can contain an arbitrary number of control messages (GO, OK, or RESEND), each with its own sequence number.

c. There are seven packet types which can travel in either direction, although the TACO2 usage of NETBLT may limit their direction of travel. These packet types either have special ways of insuring reliability, or are not transmitted reliably. They are the OPEN, RESPONSE, REFUSED, QUIT, QUITACK, DONE, and ABORT packets. In the TACO2 usage of NETBLT, the OPEN packet shall travel from sender to receiver; the RESPONSE, REFUSED, and DONE packets shall travel from receiver to sender; and the QUIT, QUITACK, and ABORT packets can be sent by both sending and receiving NETBLTs.

All packet headers shall be "longword-aligned," such as, all packet headers are a multiple of four bytes in length and all four-byte fields start on a longword boundary. The Client String field shall be terminated with at least one null byte, with extra null bytes added at the end to create a field that is a multiple of four bytes long. All numeric values shall be coded as binary integers.

5.2.7.1 <u>OPEN (type 0) and RESPONSE (type 1)</u>.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-------------------------------+---------------+---------------+
|           Checksum            |    Version    |     Type      |
+-------------------------------+---------------+---------------+
|            Length             |          Local Port           |
+-------------------------------+-------------------------------+
|         Foreign Port          |   Longword Alignment Padding  |
+-------------------------------+-------------------------------+
|                    Connection Unique ID                       |
+---------------------------------------------------------------+
|                         Buffer Size                           |
+-------------------------------+-------------------------------+
|       DATA packet size        |          Burst Size           |
+-------------------------------+-------------------------------+
|        Burst Interval         |       Death Timer Value       |
+---------------------------+-+-+-------------------------------+
|      Reserved (MBZ)       |C|M| Maximum # Outstanding Buffers |
+---------------------------+-+-+-------------------------------+
| Client String...
+-------------------------------------
                        |  Longword Alignment Padding         |
                        +-------------------------------------+
```

a. Checksum: To generate the checksum, the checksum field itself is cleared, the 16-bit ones-complement sum is computed over the packet, and the ones complement of this sum is placed in the checksum field.

b. Version: The NETBLT protocol version number

c. Type: The NETBLT packet type number (OPEN = 0, RESPONSE = 1)

d. Length: The total length (NETBLT header plus data, if present) of the NETBLT packet in bytes

e. Local Port: The local NETBLT's 16-bit port number

f. Foreign Port: The foreign NETBLT's 16-bit port number

g. Connection UID: The 32 bit connection UID specified in 5.2.5.1.6.

h. Buffer size: Size in bytes of each NETBLT buffer (except the last)

i. Data packet size: Length of each DATA packet in bytes

j. Burst Size: Number of DATA packets in a burst

k. Burst Interval: Transmit time in milliseconds of a single burst

l. Death timer: Packet sender's death timer value in seconds

m. "C": The DATA packet data checksum flag (0 = do not checksum DATA packet data, 1 = do). Shall be 1 in TACO2.

n. "M": The transfer mode (0 = READ, 1 = WRITE). (see 5.2.9.2.5).

o. Maximum # Outstanding Buffers: Maximum number of buffers that can be transferred before waiting for an OK message from the receiving NETBLT.

p. Client string: An arbitrary, null-terminated, longword-aligned string for use by NETBLT clients. Contains the metamessage in TACO2.

(NOTE: the Reserved (MBZ) field may be used as the Connection Number field by the Header Abbreviation sublayer (see 5.4.1)).

5.2.7.2 <u>QUITACK (type 3), and DONE (type 10)</u>.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Bits 0–15	Bits 16–24	Bits 25–31
Checksum	Version	Type
Length	Local Port	
Foreign Port	Longword Alignment Padding	

5.2.7.3 <u>QUIT (type 2), ABORT (type 4), and REFUSED (type 9)</u>.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Bits 0–15	Bits 16–24	Bits 25–31
Checksum	Version	Type
Length	Local Port	
Foreign Port	Longword Alignment Padding	
Reason for QUIT/ABORT/REFUSE ...		
	Longword Alignment Padding	

5.2.7.4 <u>DATA (type 5) and LDATA (type 6)</u>.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

<table>
<tr><td>Checksum</td><td>Version</td><td colspan="2">Type</td></tr>
<tr><td>Length</td><td colspan="3">Local Port</td></tr>
<tr><td>Foreign Port</td><td colspan="3">Longword Alignment Padding</td></tr>
<tr><td colspan="4">Buffer Number</td></tr>
<tr><td colspan="4">Last Buffer Touched</td></tr>
<tr><td>High Consecutive Seq Num Rcvd</td><td colspan="3">Packet Number</td></tr>
<tr><td>Data Area Checksum Value</td><td colspan="2">Reserved (MBZ)</td><td>L</td></tr>
<tr><td>New Burst Size</td><td colspan="3">New Burst Interval</td></tr>
</table>

a. Checksum: Checksum of the packet header only, including the Data Area Checksum Value.

b. Buffer number: A 32 bit unique number assigned to every buffer. Numbers are monotonically increasing, starting with 1.

c. Last Buffer Touched: The number of the highest buffer transmitted so far.

d. High Consecutive Sequence Number Received: Highest control message sequence number below which all control message sequence numbers received are consecutive.

e. Packet number: Monotonically increasing DATA packet identifier, starting with zero in each buffer.

f. Data Area Checksum Value: Checksum of the DATA packet's data. Algorithm used is the same as that used to compute checksums of other NETBLT packets.

g. "L" is a bit that is set to 1 when the buffer that this DATA packet belongs to is the last buffer in the transfer.

h. New Burst Size: Burst size as negotiated from value given by receiving NETBLT in OK message.

i. New Burst Interval: Burst interval as negotiated from value given by receiving NETBLT in OK message. Value is in milliseconds.

5.2.7.5 NULL-ACK (type 7).

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Checksum	Version	Type
Length	Local Port	
Foreign Port	Longword Alignment Padding	
Last Buffer Touched		
High Consecutive Seq Num Rcvd	New Burst Size	
New Burst Interval	Longword Alignment Padding	L

a. Last Buffer Touched: The number of the highest buffer transmitted so far.

b. High Consecutive Sequence Number Received: Same as in DATA/LDATA packet.

c. New Burst Size: Burst size as negotiated (half- and full-duplex only) from value given by receiving NETBLT in OK message.

d. New Burst Interval: Burst interval as negotiated (half- and full-duplex only) from value given by receiving NETBLT in OK message. Value is in milliseconds.

e. "L" is a bit that is set to 1 when the buffer identified in the Last Buffer Touched field is the last buffer in the transfer.

5.2.7.6 CONTROL (type 8).

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Checksum	Version	Type
Length	Local Port	
Foreign Port	Longword Alignment Padding	

Followed by any number of messages, each of which is longword aligned, with the following formats:

5.2.7.6.1 GO message (type 0).

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

<table>
<tr><td>Type</td><td>Word Padding</td><td colspan="2">Sequence Number</td></tr>
<tr><td colspan="4">Buffer Number</td></tr>
</table>

a. Type: Message type (GO = 0, OK = 1, RESEND = 2)

b. Sequence number: A 16-bit unique message number. Sequence numbers must be monotonically increasing, starting with 1.

c. Buffer number: As in DATA/LDATA packet

5.2.7.6.2 OK message (type 1).

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

<table>
<tr><td>Type</td><td>Word Padding</td><td colspan="2">Sequence Number</td></tr>
<tr><td colspan="4">Buffer Number</td></tr>
<tr><td colspan="2">New Offered Burst Size</td><td colspan="2">New Offered Burst Interval</td></tr>
<tr><td colspan="2">Current Control Timer Value</td><td colspan="2">Longword Alignment Padding</td></tr>
</table>

a. New offered burst size: Burst size for subsequent buffer transfers, possibly based on performance information for previous buffer transfers.

b. New offered burst interval: Burst interval for subsequent buffer transfers, possibly based on performance information for previous buffer transfers. Interval is in milliseconds.

c. Current control timer value: Receiving NETBLT's control timer value in milliseconds.

5.2.7.6.3 <u>RESEND message (type 2)</u>.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------------------------------+
|     Type      | Word Padding  |        Sequence Number        |
+---------------+---------------+-------------------------------+
|                         Buffer Number                         |
+-------------------------------+-------------------------------+
|  Number of Missing Packets    |    New Offered Burst Size     |
+-------------------------------+-------------------------------+
|  New Offered Burst Interval   |  Longword Alignment Padding   |
+-------------------------------+-------------------------------+
| Packet Number (2 bytes/packet)|    ...
+-------------------------------+---------+
                |         Padding (if necessary)                |
                +-----------------------------------------------+
```

a. Packet number: The 16-bit data packet identifier of a DATA packet, from the buffer identified by Buffer Number, whose retransmission is requested. Multiple packet numbers may occur in one RESEND message.

5.2.8 <u>Required NETBLT components</u>. TACO2 uses three modes of NETBLT operation; simplex, half-duplex, and full-duplex. This section identifies the required components of NETBLT for each mode of operation.

5.2.8.1 <u>Simplex</u>. The only NETBLT packet types used in the simplex case shall be the following:

a. OPEN

b. QUIT

c. ABORT

d. DATA

e. LDATA

f. NULL-ACK

5.2.8.1.1 <u>Sender simplex operation</u>. Operation of NETBLT in simplex send mode shall be as follows: the OPEN message is sent; DATA, LDATA, and possibly NULL-ACK packets are sent; and the connection is closed. Any packet may be sent more than once, for redundancy, but for all n, packets from buffer(n - 1) shall not be sent after packets from buffer(n). QUIT and ABORT packets may be sent at any time, and shall have the same effect. The Maximum Number of Outstanding Buffers (in the OPEN packet) shall be set to 2.

5.2.8.1.2 Receiver simplex operation. Operation of NETBLT in simplex receive mode shall be as follows: when an OPEN packet is received, a connection is considered to be established. Packets received shall be stored into NETBLT BUFFERS. The receiving NETBLT shall pass a buffer to the client when the buffer is filled with correct packets or when good packets for a higher-numbered buffer are received. A list of packets that are possibly bad, or missing, shall be passed to the client. When the last buffer (L flag set in packet headers) has been passed to the client, or when the death timeout has expired, the receiving connection shall be terminated.

5.2.8.1.2.1 Packet error handling. The receiving NETBLT shall discard redundant packets. In the case of errors, the following rules shall apply at the receiving NETBLT:

a. A NETBLT packet with a bad checksum shall be discarded, unless it is a DATA or LDATA packet.

b. A NETBLT DATA or LDATA packet, with a bad header checksum or data area checksum, optionally may be saved but flagged as possibly bad. Reasonableness checks shall be used to insure that good data is not affected by the possibly bad packet header or data. If a good NETBLT packet (redundantly transmitted) is received with the same buffer and packet number as a possibly bad one, the possibly bad packet shall be replaced with the good one.

5.2.8.2 Half-duplex. The normal, full-duplex version of NETBLT shall operate across half-duplex connections with the following modification: keepalive packets shall not be sent by the receiver while it is in the process of receiving a packet. The burst timer and burst size counter shall be reset at the start of each transmission period. If the Maximum Number of Outstanding Buffers (in the OPEN packet) is set to 1, the sending and receiving NETBLTs will operate in lockstep. If the Maximum Number of Outstanding Buffers is set to a value N greater than 1, the receiving NETBLT shall wait until N buffers have been completely received or have had their data timers expire before sending a CONTROL packet. An exception occurs when the last buffer is sent; when all buffers up to and including the last buffer have been completely received or have had their data timers expire, the receiving NETBLT shall be permitted to send its CONTROL packet. The last buffer is identified by the receiver as the buffer for which the "L" bit is set in a DATA/LDATA packet, or as the Last Buffer Touched in a NULL-ACK packet with its "L" bit set to 1.

NOTE: Early implementations of TACO2 did not include the "L" bit in the NULL-ACK packet. These implementations must set the Maximum Number of Outstanding Packets to 1; otherwise, loss of the entire last buffer, due to communications error, would cause the receiver to stop operating.

5.2.8.3 Full-duplex. Across full-duplex connections the normal NETBLT as described in 5.2.1 through 5.2.6 shall be used.

5.2.9 Specific values for NETBLT. The following are comments on, or specific values for, various NETBLT fields.

5.2.9.1 Fields common to all packets.

5.2.9.1.1 Version. TACO2 shall use version 4 of NETBLT.

5.2.9.1.2 Local port and foreign port. The "well-known" port, to which OPEN messages should be directed, shall be port number 1. This signifies that a connection with the TACO2 NRTS is to be established. Some other randomly-selected value shall be used for local port, this value shall be unique in that if more than one NETBLT connection is supported by a single host interface, the port number shall not be duplicated.

5.2.9.1.3 Longword alignment padding. The content of these fields shall be zeros.

5.2.9.2 OPEN and RESPONSE packets.

5.2.9.2.1 Connection UID. Connection UID may be any randomly-selected value, which shall be unique in that if more than one NETBLT connection is supported by a single host interface, it shall not be duplicated. More than one connection may be supported concurrently, to support multiplexing, but only the ability to support a single connection is required.

5.2.9.2.2 Buffer size. A TACO2 implementation shall support a buffer size of at least 4096 data bytes. In addition, it shall support any number of DATA/LDATA packets per buffer up to and including 32 packets; support for a larger number of packets per buffer is allowed but not required.

5.2.9.2.3 DATA packet size. A TACO2 implementation shall support a maximum DATA and LDATA packet size of at least 512 data bytes. DATA packets as small as 64 data bytes shall be supported. LDATA packets may be as small as one data byte, depending on the relationship between message, buffer, and packet sizes.

5.2.9.2.4 Burst size and burst interval. In point-to-point connections, burst size and burst interval ordinarily shall be set to produce the maximum data flow the connection and the hosts can support (such as, [burst_size * bytes_per_packet * bits_per_byte*1000 / burst_interval] shall be approximately equal to the apparent bits per second sent to the link). Alternatively, the burst interval may be set to zero, in which case no internal flow control shall be imposed (see 5.2.3.5).

5.2.9.2.5 Direction. (Effectivity 4) Until the effectivity date, operation of TACO2 is defined only for "M" set to 1; that is, TACO2 allows only active sending and passive receiving. Following that date, operation with "M" set to 0 is also permissible.

5.2.9.2.6 Checksumming. The value of "C" bit shall be 1; that is, TACO2 requires data checksums on DATA/LDATA packets.

5.2.9.2.7 Maximum number of outstanding buffers. A TACO2 implementation shall be capable of supporting at least two concurrently outstanding buffers.

5.2.9.2.8 Client string. This field shall contain the metamessage, which is generated by the TACO2 NRTS. Both OPEN and RESPONSE messages shall include the metamessage; a mechanism for negotiation of the value of some components is defined in the TACO2 NRTS.

5.2.9.3 QUIT packets.

5.2.9.3.1 Reason for QUIT/ABORT/REFUSE. The reason shall be an appropriate ASCII string up to 80 characters long, suitable for display to the recipient. The use of QUIT implies preemption or a probable malfunction.

(NOTE: Strings used may include:

a. "Error: unknown return from setup upcall"

b. "Error: fatal application buffer setup error"

c. "Error: unknown return from flush upcall"

d. "Error: fatal application buffer flush error"

e. "Error: fatal buffer setup error"

f. "Error: unknown return from buffer flush upcall")

5.2.9.4 ABORT packets.

5.2.9.4.1 Reason for QUIT/ABORT/REFUSE. The reason shall be an appropriate ASCII string up to 80 characters long, suitable for display to the recipient. The use of ABORT implies a serious malfunction.

(NOTE: Strings used may include:

a. "fatal buffer create error"

b. "test request"

c. "received strange burst size"

d. "received strange burst interval"

e. "Control packet buffer overflow"

f. "no memory for outbound-packet source route"

g. "attempt to establish > 1 connection per port pair")

5.2.9.5 REFUSED packets.

5.2.9.5.1 Reason for QUIT/ABORT/REFUSE. The reason shall be an appropriate ASCII string up to 80 characters long, suitable for display to the recipient. The use of REFUSED indicates that the connection cannot be completed for some reason.

(NOTE: Strings used may include:
"no service listening on port x," where x is the unacceptable port number.)

5.2.9.6 DATA and LDATA packets.

5.2.9.6.1 Packet number. The first data packet in each buffer shall be numbered 0.

5.2.9.6.2 Data area checksum value. All TACO2 DATA and LDATA packets shall be checksummed.

5.2.9.7 Timer precision. Timer precision in NETBLT shall be no worse than ± 100 milliseconds (msec).

5.2.9.8 Open timer value. The open timer shall initially be set to no less than two seconds. In half-duplex and full duplex, the value of the open timer shall be increased by two seconds after each timeout.

5.2.9.9 Quit timer value. The quit timer shall be set to no less than five seconds.

5.2.9.10 Death timer value. The death timer should be set to no less than two minutes.

5.3 Network layer - IP.

5.3.1 Overview. The DOD IP forms the network layer of the TACO2 protocol suite. IP provides a mechanism for transmitting blocks of data (datagrams) from sources to destinations, which are specified by 32-bit addresses. It is a "best-effort" mechanism, which provides no assurance that a datagram is delivered, but takes appropriate steps when possible to move a datagram toward its destination. IP is specified in Internet RFC 791, as amended by RFC 950 (IP Subnet Extension), RFC 919 (IP Broadcast Datagrams), and RFC 922(IP Broadcast Datagrams with Subnets). It usually also includes the Internet Control Message Protocol (ICMP), specified in RFC 792, which provides a mechanism for communicating control and error information between hosts and other hosts or gateways. Although ICMP is an integral part of IP, it uses the support of IP as if it (ICMP) were a higher level protocol. IP is also specified in MIL-STD-1777, which formally specifies a protocol consistent with RFC 791.

5.3.1.1 IP augmentations. As used in TACO2, IP may be augmented by the revised IP Security Option (RFC 1108), and by the Host Extensions for IP Multicasting (RFC 1112). These augmentations are not required in this version of TACO2, but they may be necessary for operation in certain environments. TACO2 supports a limited form of multicasting by allowing simplex receivers to "listen in" on simplex, half-duplex, or full-duplex transmissions. (Effectivity 5: later versions of TACO2 may support acknowledged multicast).

5.3.2 Required IP components. Because TACO2 uses IP outside its normal internetworked environment, some components of IP are unnecessary or inappropriate in some cases. This section identifies the required components for each major case.

5.3.2.1 Simplex. Simplex transmission may be used to support point-to-point or broadcast communication in TACO2. The Internet Header format shall be as specified in 5.3.3. The following fields shall be correctly filled in and interpreted for simplex operation:

a. Version

b. Internet Header Length

c. Total Length

d. Fragment Offset (must be 0)

e. Protocol (30 for NETBLT)

f. Header Checksum

g. Source Address

h. Destination Address

i. IP Security Option, if required

The remaining fields shall be disregarded by a receiver in simplex operation, but shall be provided by a transmitter for the sake of consistency. Datagrams shall not be fragmented. Subnetting support is not required. ICMP shall not be used in simplex communications.

5.3.2.2 Point-to-point duplex. Point-to-point duplex communications in TACO2 may be half-duplex or full-duplex. The Internet Header format shall be as specified in 5.3.3. The following fields shall be correctly filled in and interpreted for duplex operation:

a. Version

b. Internet Header Length

c. Total Length

d. Fragment Offset (must be 0)

e. Protocol (30 for NETBLT)

f. Header Checksum

g. Source Address

h. Destination Address

i. IP Security Option, if required

The remaining fields are not meaningful for point-to-point operation, but shall be provided by a transmitter for the sake of consistency. Datagrams shall not be fragmented. Subnetting support is not required. ICMP shall be used in point-to-point communications. However, only the following ICMP messages shall be required in this environment:

j. Parameter Problem

k. Echo

l. Echo Reply

Other ICMP messages are optional in point-to-point operation of TACO2, and shall not affect the operation of a receiver that does not implement them.

5.3.3 IP Message format for TACO2. Figure 17 is a summary of the contents of the internet header.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-------+-------+---------------+-------------------------------+
|Version|  IHL  |Type of Service|          Total Length         |
+-------+-------+---------------+-----+-------------------------+
|         Identification        |Flags|      Fragment Offset    |
+---------------+---------------+-----+-------------------------+
|  Time to Live |    Protocol   |         Header Checksum       |
+---------------+---------------+-------------------------------+
|                       Source Address                          |
+---------------------------------------------------------------+
|                    Destination Address                        |
+---------------------------------------------------------------+
|                       Options (if any)                        |
+-----------------------------------------------+---------------+
|                     . . .                     |Padding (if any)|
+-----------------------------------------------+---------------+
```

FIGURE 17. Internet datagram header.

a. Version: 4 bits
The Version field indicates the format of the internet header. This value shall be 4.

b. IHL: 4 bits
Internet Header Length (IHL) is the length of the internet header in 32-bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

c. Type of Service: 8 bits
Type of service choices are not meaningful in point-to-point and simplex operation. In TACO2, this field shall normally have value 0 (routine precedence, normal delay, normal throughput, normal reliability).

d. Total Length: 16 bits
Total Length is the length of the datagram, measured in octets, including internet header and data. All hosts shall be prepared to accept datagrams of up to 576 octets.

e. Identification: 16 bits
An identifying value assigned by the sender; may be ignored in TACO2.

f. Flags: 3 bits, Fragment Offset: 13 bits
TACO2 packets shall not be fragmented. A packet size shall be used that makes fragmentation unnecessary.

g. Time to Live: 8 bits
May be ignored in TACO2. This field indicates the maximum time the datagram is allowed to remain in an internet system. If this field contains the value zero, the datagram shall not be forwarded to another node.

h. Protocol: 8 bits
This field indicates the next level protocol used in the data portion of the internet datagram. The next higher level protocol used in TACO2 shall be NETBLT, which has been assigned number 30 (decimal). ICMP is protocol number 1.

i. Header Checksum: 16 bits
A checksum on the header only. The checksum field's value shall be the 16-bit one's complement of the one's complement sum of all 16-bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero. A received datagram with an incorrect header checksum shall be discarded.

j. Source Address: 32 bits
The 32-bit IP-style address of the datagram's source. For point-to-point use, this address may be assigned arbitrarily, but shall be consistent with normal IP usage; in particular, the host portion of the address shall be neither all 1's nor all 0's (binary).

k. Destination Address: 32 bits
The 32-bit IP-style address of the datagram's destination. For point-to-point use, this address may be assigned arbitrarily, but shall be consistent with normal IP usage; in particular, the host portion of the address shall not be all 0's, and a host portion of all 1's shall be interpreted as a broadcast address. Multicast addressing per RFC 1112 may be incorporated, but is not required in this version of TACO2.

l. Options: variable
The options are optional in each datagram. If the options do not end on a 32-bit boundary, the internet header shall be filled out with octets of zeros. The first of these shall be interpreted as the end-of-options option, and the remainder as internet header padding. For the purpose of TACO2 end-points, options other than security should not be generated, but shall be tolerated if received.

5.3.4 ICMP.

5.3.4.1 Overview. Occasionally a destination host will communicate with a source host, for example to report an error in datagram processing. For such purposes the Internet Control Message Protocol (ICMP), shall be used. ICMP uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP. ICMP messages typically report errors in the processing of datagrams. To avoid the infinite loop of messages about messages, no ICMP messages shall be sent about ICMP messages.

5.3.4.2 ICMP in TACO2. The message formats described here are for the ICMP messages required for point-to-point duplex communications with TACO2 (see 5.3.2.2).

5.3.4.3 ICMP message formats. ICMP messages are sent using the basic IP header. The first octet of the data portion of the datagram is an ICMP type field; the value of this field determines the format of the remaining data. Any field labeled "unused" is reserved for later extensions and shall be zero when sent, but receivers shall not use these fields (except to include them in the checksum). The values of the following internet header fields shall be as described in 5.3.3:

a. Version

b. IHL

c. Type of Service

d. Total Length

e. Identification

f. Flags

g. Fragment Offset

h. Time to Live

i. Header Checksum

j. Source Address

k. Destination Address

The Protocol field shall have value 1 for ICMP.

5.3.4.3.1 Parameter problem message.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Bits 0–7	Bits 8–15	Bits 16–31
Type	Code	Checksum
Pointer	Unused	
Internet Header + 64 bits of Original Data Datagram		

5.3.4.3.1.1 IP fields.

a. Destination Address: The source network and address from the original datagram's data.

5.3.4.3.1.2 ICMP fields.

a. Type: 12

b. Code: 0, indicating that the pointer field indicates the error.

c. Checksum: The checksum shall be the 16-bit one's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field shall be zero.

d. Pointer: identifies the octet where an error was detected.

e. Internet Header + 64 bits of Data Datagram: The internet header plus the first 64 bits of the original datagram's data. This data may be used by the host to match the message to the appropriate process.

If the host processing a datagram finds a problem with the header parameters, so that it cannot complete processing the datagram, it shall discard the datagram. The host also may notify the source host via the parameter problem message. This message shall be sent only if the error caused the datagram to be discarded. The pointer shall identify the octet of the original datagram's header where the error was detected (it may be in the middle of an option). For example, 1 indicates something is wrong with the Type of Service, and (if there are options present) 20 indicates something is wrong with the type code of the first option.

5.3.4.3.2 Echo or echo reply message.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Type	Code	Checksum
Identifier		Sequence Number
Data...		

5.3.4.3.2.1 IP fields.

a. Addresses: The address of the source in an echo message shall be the destination of the echo reply message. To form an echo reply message, the source and destination addresses are simply reversed, the type code changed to 0, and the checksum recomputed.

5.3.4.3.2.2 ICMP fields.

a. Type: 8, indicating echo message, or 0 for echo reply message.

b. Code: 0

c. Checksum: The checksum shall be the 16-bit one's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field shall be zero. If the total length is odd, the received data is padded with one octet of zeros for computing the checksum.

d. Identifier: an identifier to aid in matching echoes and replies, may be zero.

e. Sequence Number: a sequence number to aid in matching echoes and replies, may be zero. The data received in the echo message shall be returned in the echo reply message. The identifier and sequence number may be used by the echo sender to help match the replies with the echo requests. For example, the identifier might be used like a port in Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) to identify a session, and the sequence number might be incremented on each echo request sent. The echoer returns these same values in the echo reply.

5.4 Data link layer. The Data Link layer in TACO2 is divided into three sublayers: Header Abbreviation, FEC, and Framing. (Effectivity 6: a Medium Access Control Layer, just below the Framing Sublayer, is under consideration.)

5.4.1 Header abbreviation sublayer. TACO2 provides a mechanism for header abbreviation across point-to-point links. Using the header abbreviation sublayer is optional; its inclusion in any compliant implementation of TACO2 shall be mandatory (Effectivity 3). The normal size of a combined NETBLT/IP header for a DATA or LDATA packet is 52 bytes. The overhead due to this header size is significant when small packets are used. TACO2 therefore provides an option for DATA/LDATA packet header abbreviation that reduces the size of headers to eight bytes. The abbreviation mechanism takes advantage of the fact that some header fields are generous in size, and others rarely or never change. A header abbreviated by the sender, in conjunction with state information stored by the receiver, provides all the information necessary to reconstruct the original header at the receiver.

5.4.1.1 Abbreviated header format. The format of a TACO2 packet with abbreviated header is illustrated on figure 18.

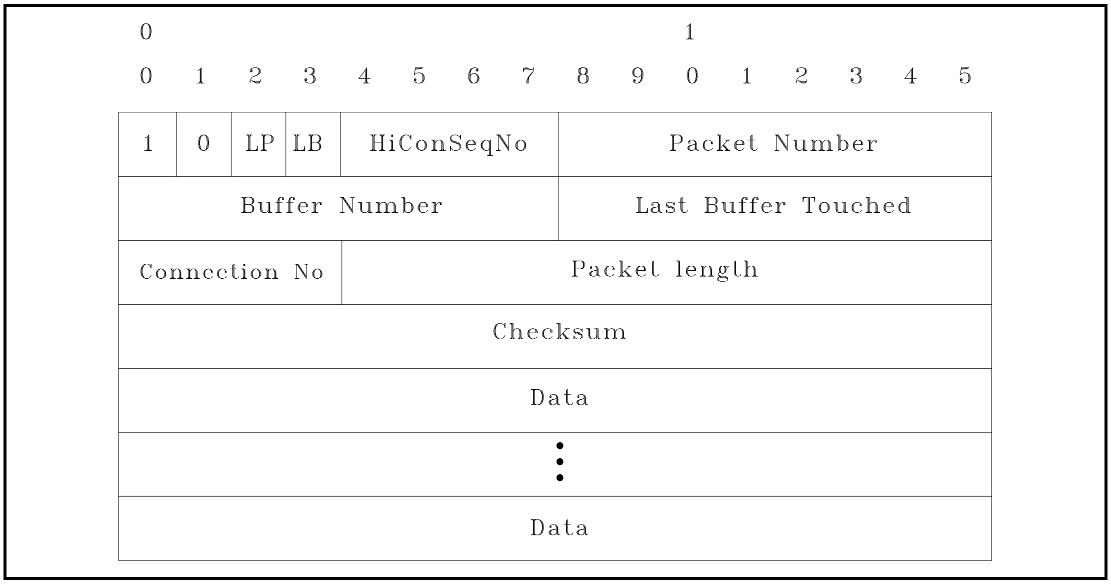

FIGURE 18. Abbreviated header TACO2 packet.

The first two bits of the packet are 10. The remaining fields are defined as follows:

a. LP: if this bit is 1, this is the last packet in a buffer (such as, a abbreviated LDATA packet).

b. LB: if this bit is 1, the buffer that this packet belongs to is the last buffer in a transfer.

c. HiConSeqNo: the low-order four bits of the High Consecutive Sequence Number field from the DATA/LDATA packet.

d. Packet Number: the low-order eight bits of the Packet Number field from the DATA/LDATA packet.

e. Buffer Number: the low-order eight bits of the Buffer Number field from the DATA/LDATA packet.

f. Last Buffer Touched: the low-order eight bits of the Last Buffer Touched field from the DATA/LDATA packet.

g. Connection No: the number of the connection to which this packet belongs. In TACO2 implementations that support only one connection at a time, this field shall be 0.

h. Packet Length: the total length (header plus data) of the abbreviated packet in bytes.

i. Checksum: to generate the checksum, the checksum field itself is cleared, the 16-bit ones-complement sum is computed over the abbreviated packet, and the ones complement of this sum is placed in the checksum field (see 5.2.4 for a discussion of checksumming).

j. Data: the Data portion of the original DATA/LDATA packet.

5.4.1.2 Multiple-connection operation with abbreviated headers. TACO2 provides two compatible mechanisms for operation with abbreviated headers. The first, described in this section, allows multiple connections to operate concurrently across a single point-to-point link. The second, described in 5.4.1.3, allows only a single connection at a time across a point-to-point link.

5.4.1.2.1 Sender operation with abbreviated headers. A sender shall provide the option of operation with or without abbreviated headers. If operation with abbreviated headers is selected, the sender shall abbreviate DATA/LDATA packets for an open connection.

5.4.1.2.1.1 Sender connection state table. A sender shall maintain the following information about each connection. The following description, in terms of indexing state table entries, is not intended to dictate an implementation. It describes operation in terms of a connection state table, with space for 16 entries. Each entry includes fields for the following:

a. Source IP address.

b. Destination IP address.

c. Local Port number.

d. Foreign Port number.

e. Connection UID.

f. Burst Size.

g. Burst Interval.

The index of the entry with matching field values shall be used as the abbreviated packet's Connection Number, as explained below.

5.4.1.2.1.2 Sender processing of outgoing OPEN packet. When a sender with header abbreviation turned on detects a NETBLT OPEN packet, it shall examine its connection state table for an entry with the same IP addresses, Local Port number, and Connection UID. If no such entry is found, it shall establish a table entry with those values, using the least recently used entry if no unused entries are available. (Note that reuse of the state table entry for an open connection will make it impossible to abbreviate any more packets for that connection.) The index of the entry with the same IP addresses, Local Port number, and Connection UID shall be used as the connection number. The connection number shall be inserted into the high-order four bits of the Reserved field (referred to in the following sections as the Connection Number field for OPEN and RESPONSE packets) in the OPEN packet, and the packet sent on to the next lower layer. The modified OPEN packet is shown below.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-------------------------------+-----------------+-------------+
|            Checksum           |     Version     |    Type     |
+-------------------------------+-----------------+-------------+
|             Length            |           Local Port          |
+-------------------------------+-------------------------------+
|          Foreign Port         |  Longword Alignment Padding   |
+-------------------------------+-------------------------------+
|                     Connection Unique ID                      |
+---------------------------------------------------------------+
|                          Buffer Size                          |
+-------------------------------+-------------------------------+
|        DATA packet size       |           Burst Size          |
+-------------------------------+-------------------------------+
|         Burst Interval        |       Death Timer Value       |
+-------+---------------------+-+-+-----------------------------+
|Conn # |   Reserved (MBZ)    |C|M| Maximum # Outstanding Buffers|
+-------+---------------------+-+-+-----------------------------+
| Client String...                    
+-------------------------------------
                           Longword Alignment Padding           |
                          --------------------------------------+
```

5.4.1.2.1.3 Sender processing of incoming RESPONSE packet. When a sender with header abbreviation turned on detects a NETBLT RESPONSE packet, it shall examine the entry in its connection state table identified by the value of the Connection Number field for IP source and destination addresses equal to the packet's destination and source addresses, Local Port number equal to the packet's Foreign Port number, and the same Connection UID. If the values match, the Local Port number, burst size, and burst interval in the RESPONSE packet shall replace the Foreign Port number, burst size, and burst interval in the table entry. The packet shall be sent on to the IP layer.

5.4.1.2.1.4 Sender processing of outgoing DATA/LDATA packet. When a sender with header abbreviation turned on detects a NETBLT DATA/LDATA packet, it shall examine the connection state table for an entry with the same IP addresses, port numbers, Burst Size, and Burst Interval. If such an entry is found, it shall abbreviate the packet using the format defined in 5.4.1.1, using the index of the entry as the Connection Number, and send it on to the next lower layer. If no such entry is found, the packet shall be sent on unchanged. If a connection state table entry is found with the same IP addresses and port numbers, the burst size and burst interval values in the packet shall replace those in the table entry.

5.4.1.2.2 Receiver operation with abbreviated headers. The receiver shall check the high-order two bits of each received packet to determine the packet type. If the bits are 01, the packet shall be passed unchanged to the IP layer. If the bits are 10, the packet shall be reconstructed by the Header Abbreviation sublayer; the Header Abbreviation sublayer shall pass the reconstructed packet to the IP layer.

5.4.1.2.2.1 Receiver connection state table. The receiver shall maintain a connection state table, with space for 16 entries, containing information needed for correct reconstruction of the reconstructed packet. Each entry shall include fields for the following:

a. Source IP address.

b. Destination IP address.

c. Local Port number.

d. Foreign Port number.

e. Connection UID.

f. Burst Size.

g. Burst Interval.

h. Buffer Number.

i. Last Buffer Touched.

j. High Consecutive Sequence Number Received.

The abbreviated packet's Connection Number shall be used as an index into the connection state table, as explained below.

5.4.1.2.2.2 Receiver processing of incoming OPEN packet. When a receiver detects a NETBLT OPEN packet, it shall fill in the IP address, Local Port number, and Connection UID fields in the connection state table entry indexed by the packet's Connection Number with the values in the packet.

5.4.1.2.2.3 Receiver processing of outgoing RESPONSE packet. When a receiver detects a NETBLT RESPONSE packet, it shall examine its connection state table for an entry with IP source and destination addresses equal to the packet's destination and source addresses, Local Port number equal to the packet's Foreign Port number, and the same Connection UID. If the values match, the Local Port number, burst size, and burst interval in the RESPONSE packet shall replace the Foreign Port number, burst size, and burst interval in the table entry, and the Connection Number field in the RESPONSE packet shall be filled in with the index of the matching connection state table entry.

5.4.1.2.2.4 Receiver processing of incoming abbreviated header packet. When a receiver detects a packet with abbreviated header, it shall use the values in the connection state table entry indexed by the Connection Number to determine the IP addresses, port numbers, burst size, and burst interval in the reconstructed packet. The remaining IP header fields shall be filled in accordance with 5.3.3. The NETBLT Buffer Number, Last Buffer Touched, and High Consecutive Sequence Number Received fields shall be filled in with values computed by combining the abbreviated header value of the corresponding field and the receiver's state table value. For the Last Buffer Touched and High Consecutive Sequence Number Received fields, the computed value shall contain the abbreviated header value for the low-order portion, and the smallest value such that the result is no less than the old state table value for the high-order portion. For the Buffer Number field, the computed value shall use the abbreviated header value for the low-order portion, and a value that causes minimal change between the old state table value and the new computed value for the high-order portion. The state table value shall be changed to correspond to the new computed value. The NETBLT Packet Number field shall be filled in with the abbreviated header Packet Number field, padded with zeros on the left. The NETBLT "L" bit shall be set according to the "LB" bit in the abbreviated header, and the NETBLT Type field shall be DATA or LDATA according to the "LP" bit in the abbreviated header. The remaining NETBLT fields shall be filled in accordance with 5.2.

5.4.1.2.2.5 Receiver processing of incoming DATA/LDATA packet. When a receiver detects a NETBLT DATA/LDATA packet, it shall examine the connection state table for an entry with the same IP addresses and port numbers. If such an entry is found, it shall replace the burst size and burst interval fields in that entry with the values from the DATA/LDATA packet. The packet shall be sent to the IP layer unchanged.

5.4.1.3 Single-connection operation with abbreviated headers. In single-connection operation of TACO2 with abbreviated headers, the connection number shall be ignored. Operation shall be as described in 5.4.1.2, except that the connection state table shall contain only one entry, with the index zero. However, since TACO2 does not mandate interlayer interfaces, it may be implemented in a simpler way, as described in the following.

5.4.1.3.1 Single-connection sender operation with abbreviated headers. The sender shall provide the option of operation with or without abbreviated headers. If operation with abbreviated headers is selected, the sender shall abbreviate DATA/LDATA. The sender shall maintain a set of connection state variables for the following:

a. Burst Size.

b. Burst Interval.

5.4.1.3.1.1 Sender processing of outgoing OPEN packet. In single-connection operation, the sender with header abbreviation turned on shall store the Burst Size and Burst Interval in the connection state variables and pass the OPEN packet unchanged to the next lower layer.

5.4.1.3.1.2 Sender processing of incoming RESPONSE packet. In single-connection operation, the sender with header abbreviation turned on shall store the Burst Size and Burst Interval in the connection state variables and pass RESPONSE packet unchanged to the next higher layer.

5.4.1.3.1.3 Sender processing of outgoing DATA and LDATA packets. In single-connection operation, the sender with header abbreviation turned on shall compare the DATA/LDATA packet's Burst Size and Burst Interval with the values stored in the connection state variables. If both match, it shall abbreviate the packet using the format defined in 5.4.1.1, using zero as the Connection Number. If either variable does not match, the DATA/LDATA packet shall be sent on unchanged, and the burst size and burst interval values in the packet shall replace those in the table entry.

5.4.1.3.2 Single-connection receiver operation with abbreviated headers. The receiver shall check the high-order two bits of each received packet to determine the packet type. If the bits are 01, the packet shall be passed unchanged to the IP layer. If the bits are 10, the packet shall be reconstructed by the Header Abbreviation sublayer ; the Header Abbreviation sublayer shall pass the reconstructed packet to the IP layer. The receiver shall maintain a set of connection state variables, corresponding in size to those in a DATA/LDATA packet, for the following:

a. Buffer Number.

b. Last Buffer Touched.

c. High Consecutive Sequence Number Received.

d. Burst Size.

e. Burst Interval.

f. Local Port.

g. Foreign Port.

5.4.1.3.2.1 Receiver processing of incoming OPEN packet. In single-connection operation, upon detection of an OPEN packet, the receiver shall set the first three connection state variables to zero, store the Burst Size, Burst Interval, Local Port, and Foreign Port, and pass the OPEN packet unchanged to the IP layer.

5.4.1.3.2.2 Receiver processing of outgoing RESPONSE packets. In single-connection operation, the receiver shall store the Burst Size, Burst Interval, Local Port in the Foreign Port variable, and Foreign Port in the Local Port variable, and pass the packet unchanged to the next lower layer.

5.4.1.3.2.3 Receiver processing of incoming abbreviated-header packets. In single-connection operation, the receiver shall reconstruct a DATA/LDATA packet from the incoming abbreviated-header packet . The NETBLT Buffer Number, Last Buffer Touched, and High Consecutive Sequence Number

Received fields shall be filled in with values computed by combining the abbreviated header value of the corresponding field and the receiver's connection state variable value. For the Last Buffer Touched and High Consecutive Sequence Number Received fields, the computed value shall contain the abbreviated header value for the low-order portion, and the smallest value such that the result is no less than the old connection state variable value for the high-order portion. For the Buffer Number field, the computed value shall use the abbreviated header value for the low-order portion, and a value that causes minimal change between the old connection state variable value and the new computed value for the high-order portion. The connection state variable value shall be changed to correspond to the new computed value. The NETBLT Packet Number field shall be filled in with the abbreviated header Packet Number field, padded with zeros on the left. The NETBLT "L" bit shall be set according to the "LB" bit in the abbreviated header, and the NETBLT Type field shall be DATA or LDATA according to the "LP" bit in the abbreviated header. Burst Size and Burst Interval shall be filled in according to the variable values. The remaining NETBLT fields shall be filled in accordance with 5.2. The reconstructed NETBLT packet shall be passed directly to the NETBLT layer (bypassing the IP layer) for normal processing.

5.4.2 FEC sublayer. Using the FEC sublayer is optional; the inclusion of FEC-I in any compliant implementation of TACO2 is mandatory.

5.4.2.1 FEC-I code. The FEC-I encoding process takes each IP datagram to be transmitted, adds Reed-Solomon redundancy, and passes the encoded datagram to the link layer for encapsulation, generally as an HDLC frame as specified in 5.4.3.1. As a result, the HDLC implementation shall (TBR) allow received packets to be processed by the FEC sublayer even if the HDLC checksum is in error. The encoded datagram also may be encapsulated as a SLIP frame, as specified in 5.4.3.2. If the unencoded datagram contains K bytes where K is not greater than 152, the encoded datagram contains a single Reed-Solomon codeword containing K + 10 bytes. For purposes of Reed-Solomon code definition, these bytes are numbered from 0 to K + 9 as shown on figure 19 (where the left end of the figure represents the beginning of the datagram if viewed in time-sequence).

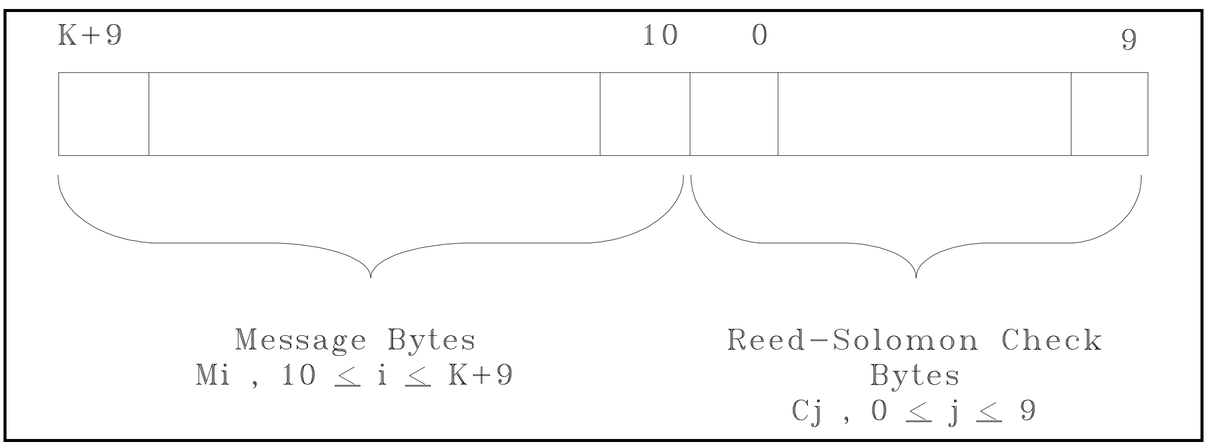

FIGURE 19. FEC-I format.

The FEC-I code uses arithmetic in the Galois field GF(256), as specified in the standard ISO 9171 for 5.25" WORM disk coding. This field has 256 elements, which are represented by 8-bit symbols ("octets" or simply "bytes"), using the generator polynomial $x^8 + x^5 + x^3 + x^2 + 1$. The primitive element a has hexadecimal value 0x02. The Reed-Solomon check bytes Cj are defined by the following:

$$C_j = \sum_{i=10}^{K+9} \frac{M_i}{a^{i} = a^{j}} \quad , \quad j = 0 \ldots 9$$

5.4.2.1.1 <u>Correction capability</u>. Each codeword as defined in 5.4.2.1 has distance 11 and is fully correctable with up to five independent byte-errors. FEC-I can fully correct all patterns of five or fewer erroneous bytes in any codeword. Note that the content and sequence of the message bytes M_i remains unchanged by the encoding process.

5.4.2.1.2 <u>Long datagrams</u>. If the length of the datagram to be encoded is greater than 152 bytes but does not exceed 752 bytes, encoding is performed by including up to five separate concatenated Reed-Solomon codewords in the encoded datagram. The maximum encoded length is 802 bytes. As an example, figure 20 shows how a datagram with an unencoded length of 450 bytes would be encoded, giving a datagram 480 bytes long.

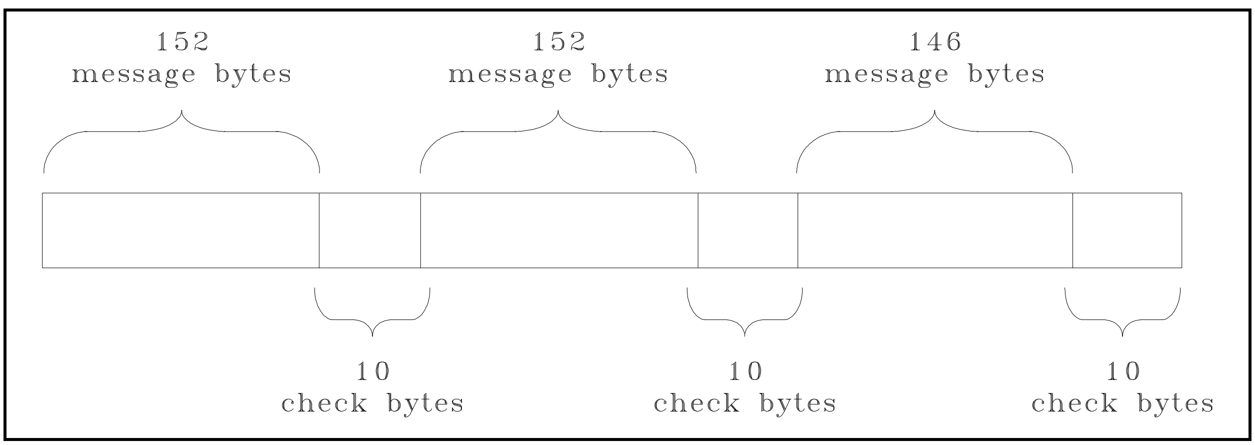

FIGURE 20. <u>Encoding a 450 byte packet</u>.

As shown on figure 20, only the final codeword in a multicodeword encoded datagram may be truncated to fewer than 152 message bytes. The time sequence of the message bytes is unchanged by the encoding process. FEC-I encoding is presently not specified for datagrams whose unencoded length is greater than 752 bytes. Should a FEC-I encoder be presented with such a datagram, the correct action is to transmit it without any encoding.

5.4.2.2 Required modes of FEC. The following mandatory FEC modes (see 5.4.2.2.1 and 5.4.2.2.2) shall be supplied by Secondary Imagery Dissemination (SID) devices. Each of these three modes shall be able to bypass the FEC capability contained in any datalink hardware external to the SID device. The operational selection of the three modes (see 5.4.2.2.1, 5.4.2.2.2, and 5.4.2.2.3) shall be controllable using the SIDS user interface.

5.4.2.2.1 Uncoded. The SLIP and/or HDLC encapsulated datalink is provided with no FEC coding applied by the SIDS system.

5.4.2.2.2 FEC-I. FEC-1 is applied to a SLIP and/or HDLC encapsulated datalink as described in 5.4.2.1.

5.4.2.2.3 FEC-II. FEC-II is applied to a SLIP and/or HDLC encapsulated datalink as described in appendix C (Effectivity 2).

5.4.2.3 Bit error ratio testing (BERT).

5.4.2.3.1 Bit error ratio test facility. The Data Link Layer shall provide the upper protocol layers with a Bit Error Ratio Test facility. The services provided to the upper layer are:

a. The ability to send a group of BERT frames over the link.

b. The ability to notify that a group of BERT frames has been received, along with the following information: number of frames successfully received; the IP address of the station originating the BERT test; and whether the received frames form a standard BERT test or a short BERT test as defined below.

5.4.2.3.2 BERT frame format. The BERT frame may be encapsulated by either the HDLC protocol or the SLIP protocol. If encapsulated by HDLC, the BERT frame shall contain 12 data bytes followed by the 2-byte frame check sequence as defined by the HDLC protocol. If encapsulated by SLIP, the BERT frame shall contain only the 12 data bytes. In either case, the 12 data bytes, numbered 0 through 11 in time sequence, shall be formatted as shown on figure 21.

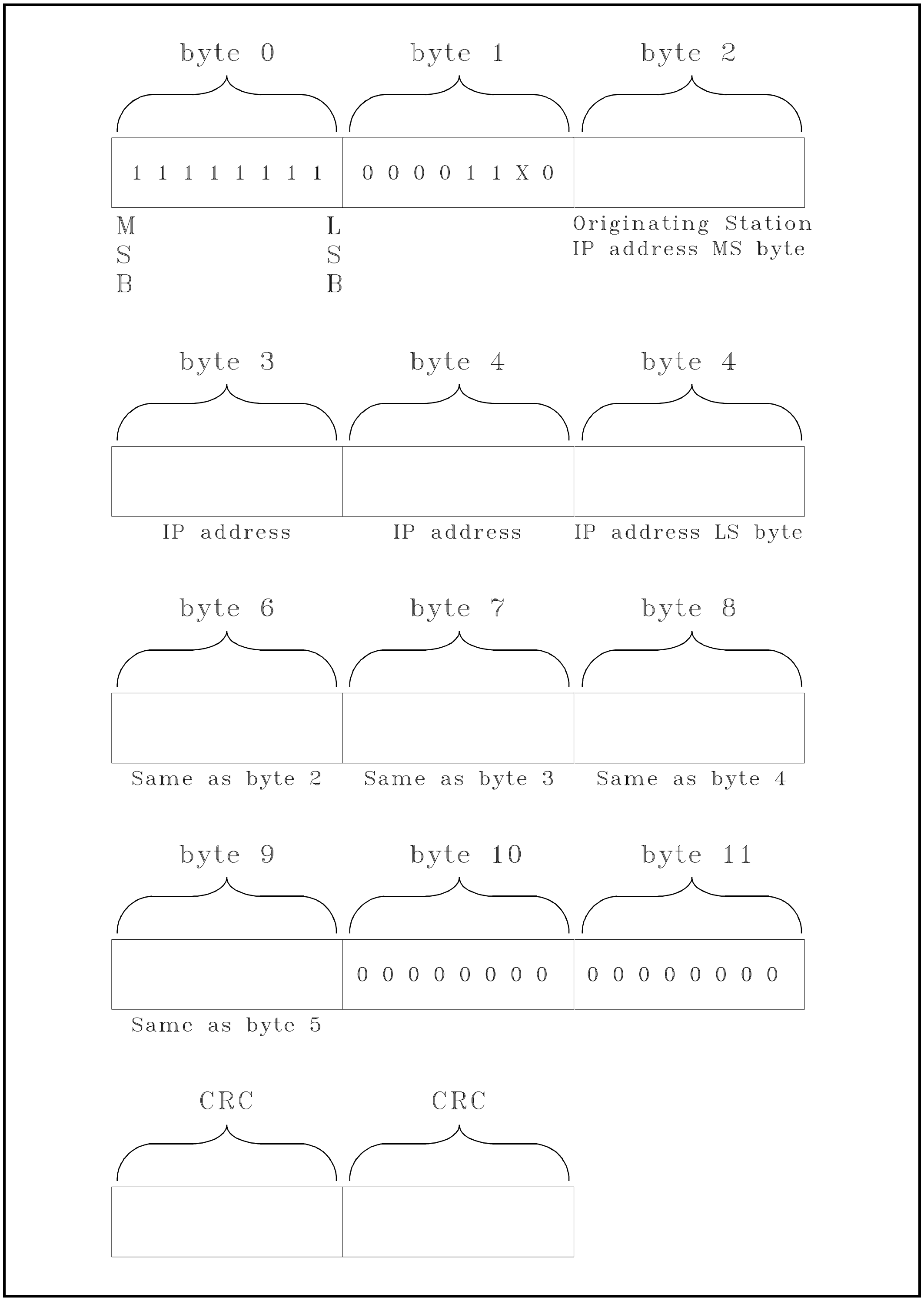

FIGURE 21. BERT frame format.

5.4.2.3.3 Standard BERT test format. The standard BERT test shall consist of sending 1,000 (one thousand) identical BERT frames, each with the hexadecimal character "0x0E" in byte number 1.

5.4.2.3.4 Short BERT test format. The short BERT test shall consist of sending 200 (two hundred) identical BERT frames, each with the hexadecimal character "0x0C" in byte number 1.

5.4.3 Framing sublayer. Two Framing sublayers are included in the TACO2 protocol stack: HDLC Framing for synchronous operation, and SLIP for asynchronous operation. HDLC Framing capability shall be available in any compliant TACO2 implementation; the implementation of SLIP is optional.

5.4.3.1 HDLC framing.

5.4.3.1.1 Overview. The synchronous data link layer for TACO2 uses HDLC framing as a standard transparent encapsulation for the IP packets supplied by the next higher protocol layer. TACO2 uses only the frame structure, and not the control procedures, of HDLC, the ISO High-level Data Link Control. The frame structure used is compatible with ISO 3309-1979 and with the International Telegraph and Telephone Consultative Committee (CCITT) Recommendation X.25, which is based on HDLC.

5.4.3.1.2 Required HDLC components. The standard HDLC frame structure is shown below. The fields shall be transmitted from left to right.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+----------------+----------------+----------------+----------------
|      Flag      |    Address     |    Control     | Information...
+----------------+----------------+----------------+-----------
                                ...
        ------------------------------------------------------------+
  ...Information |     Frame Check Sequence        |      Flag      |
-----------------+---------------------------------+----------------+
```

5.4.3.1.2.1 Flag sequence. The flag sequence shall be a single octet that indicates the beginning or end of a frame. The flag sequence consists of the binary sequence 01111110 (hexadecimal 0x7E). A flag that terminates one frame may also signal the beginning of the next frame, which allows back-to-back frames to be separated by a single flag.

5.4.3.1.2.2 Address field. The address field shall be a single octet. It shall contain 0xFF if no FEC coding is applied by the FEC sublayer, and shall contain 0x33 if FEC-1 coding, as specified in 5.4.2.1, is applied.

5.4.3.1.2.3 Control field. The control field shall be a single octet that indicates the type of frame. The control field for TACO2 shall contain the binary sequence 00000011 (hexadecimal 0x03), the Unnumbered Information (UI) command with the Poll/Final (P/F) bit set to zero.

5.4.3.1.2.4 Information field. The information field contains data for the next higher protocol layer, which for TACO2 shall be an IP packet. The end of the information field is found by locating the closing flag and allowing two octets for the Frame Check Sequence field. The Information field shall be an integer number of octets in length.

5.4.3.1.2.5 Frame check sequence field. The Frame Check Sequence (FCS) field is two octets. It shall be calculated over all bits of the address, control, and information fields, not including any bits inserted for transparency. This does not include the flag sequences or the FCS field. The polynomial used is $x^{16} + x^{12} + x^5 + 1$. This is the standard CCITT Cyclic Redundancy Check (CRC) polynomial.

5.4.3.1.3 HDLC procedures.

5.4.3.1.3.1 Order of bit transmission. The information field shall be transmitted with the high-order (most significant) octet first. Individual octets of all fields except the FCS, but including the Information field, shall be transmitted low-order bit first. The FCS shall be transmitted with the coefficient of the highest term first.

5.4.3.1.3.2 Transparency. Bit-stuffing is used to distinguish the flag sequence from other fields. The transmitter shall insert a 0 bit after all non-flag sequences of 5 contiguous 1 bits. The receiver shall discard a 0 bit received after five contiguous 1 bits. The reception of a sixth 1 bit indicates a flag when followed by a zero bit, or an abort when followed by a seventh 1 bit.

5.4.3.1.3.3 Invalid frames. In duplex mode with the optional FEC sublayer not operating, invalid frames shall be discarded. An invalid frame is one that is too short, too long, contains a nonintegral number of octets, contains an unrecognized address or control field, has an invalid FCS, or was aborted. In simplex mode and in duplex mode with the FEC sublayer operating, a frame with an invalid FCS that is otherwise valid may be passed on to the FEC sublayer. A frame of less than four octets (excluding flags) is too short; a TACO2 implementation shall support a maximum frame size of at least 576 information octets.

5.4.3.1.3.4 Frame abortion. A frame may be aborted by transmitting at least 7 contiguous 1 bits.

5.4.3.1.3.5 Inter-frame time fill. Inter-frame time fill, when required within a period of continuous transmission, shall be accomplished by transmitting flag sequences. The transmitted level for the interval between periods of continuous transmission shall be in accordance with the requirements of the attached cryptographic device, if any.

5.4.3.2 SLIP. Devices using asynchronous communications shall use SLIP.

5.4.3.2.1 Overview. The asynchronous data link layer for TACO2 uses SLIP as a standard transparent encapsulation for the IP packets supplied by the next higher protocol layer. SLIP defines a sequence of characters that frame packets on a serial line in a simple and consistent manner.

5.4.3.2.2 Protocol. The SLIP protocol defines two special characters: END and ESC. END is hexadecimal 0xC0 (decimal 192) and ESC is hexadecimal 0xDB (decimal 219), not to be confused with the ASCII ESC character. For this discussion, ESC will indicate the SLIP ESC character. To send a packet, a SLIP host shall send an END character followed by the data in the packet. If a data byte is the same code as the END character, a two-byte sequence of ESC and hexadecimal 0xDC (decimal 220) shall be sent instead. If a data byte is the same code as the ESC character, a two byte sequence of ESC and hexadecimal 0xDD (decimal 221) shall be sent instead. When the last byte in the packet has been sent, an END character shall be transmitted.

5.4.3.2.3 Required SLIP components. The only required SLIP components are the mechanisms for indicating End of Packet and for byte-stuffing.

5.4.3.2.4 Specific values for SLIP.

5.4.3.2.4.1 Order of bit transmission. Information shall be transmitted with the high-order (most significant) octet first. Individual octets shall be transferred low-order bit first. This is the standard asynchronous transmission order.

5.4.3.2.4.2 Transparency. Byte-stuffing shall be used to distinguish the End of Packet character from the same value when it occurs in the information being transmitted.

5.4.3.2.4.3 Invalid frames. The only invalid frames in SLIP shall be ones that are too long. The receiver shall be prepared to accept frames of at least 576 octets (not including framing and byte-stuffing characters), and preferably of at least 2048 octets. Frames that are too long may be discarded or truncated.

5.4.3.2.4.4 Inter-frame gap. Frames may be separated by a single END character. Any inter-frame gap shall consist of continuous marks or continuous END characters.

5.5 DTE-DCE interfaces. TACO2 does not specify a single standard Data Terminal Equipment - Data Circuit-terminating Equipment (DTE-DCE) interface. Detailed implementation information for certain communications environments (such as, recommended parameter values, interfaces, and settings/strappings for KY-57, KG-84A/C, KY-68 and STU-III cryptographic devices) is provided by the JIEO TIS listed in section 2. Additional TIS documents will be prepared for other communications environments. TACO2 implementations shall provide DTE-DCE interfaces capable of conforming to the TIS recommendations for each cryptographic device or communications environment identified by the procuring activity.

6. NOTES

(This section contains general or explanatory information that may be helpful but is not mandatory).

6.1 Example TACO2 packet. Figure 22 is an example of a TACO2 data packet, including IP and NETBLT headers, with values shown in decimal.

0 0 1 2 3 4 5 6 7	1 8 9 0 1 2 3 4 5	2 6 7 8 9 0 1 2 3	3 4 5 6 7 8 9 0 1
Ver= 4, IHL= 5	Type of Service	Total Length = 70	
Identification = 4		Flg=0	Fragment Offset = 0
Time to Live=255	Protocol = 30	Header Checksum	
Source Address = 128.83.8.2			
Destination Address = 129.83.2.81			
Checksum		Version = 4	Type = 6
Length = 50		Local Port = 21835	
Foreign Port = 1		Longword Alignment Padding = 0	
Buffer Number = 1			
Last Buffer Touched = 1			
High Cons Seq Num Rcvd = 0		Packet Number = 0	
Data Area Checksum Value		Reserved (MBZ)	1
New Burst Size = 7		New Burst Interval = 7000	
T	h	e	(Space)
M	I	T	R
E	(Space)	C	o
r	p	.	(Cr)
(Lf)	(Ctrl−z)		

FIGURE 22. Example TACO2 packet.

The first four bits transmitted would be 0100, or decimal 4; the next four bits would be 0101, or decimal 5; that is, the values, not the field identifiers, are transmitted.

This is an internet datagram in version 4 of the internet protocol; the internet header consists of five 32 bit words, and the total length of the datagram is 70 octets, where 18 is the number of data bytes in the packet. This datagram is a complete datagram (not a fragment). Following the IP header is the NETBLT header. It indicates that this is an LDATA (type 6) packet, in version 4 of NETBLT, with 50 bytes of NETBLT header plus data. It is the first packet (packet numbers start with 0) of the first buffer, and it is both the last packet in this buffer and the last buffer in this transmission. The burst size is now seven packets, and the burst interval is seven seconds.

The actual hexadecimal values transmitted for this packet would be as follows, in octet transmission order left-to-right, top-to-bottom:

```
45 00 00 46 00 04 00 00 FF 1E AF 9C 80 53 08 02
81 53 02 51 D6 A7 04 06 00 32 55 4B 00 01 00 00
00 00 00 01 00 00 00 01 00 00 00 00 71 B4 00 01
00 07 1B 58 54 68 65 20 4D 49 54 52 45 20 43 6F
72 70 2E 0D 0A 1A
```

6.2 TACO2 NETBLT compared to RFC998 NETBLT. NETBLT as specified in Internet RFC 998 is modified in this document for use as an element of TACO2. The modifications are as follows:

a. The Transfer Size field is removed from the OPEN and RESPONSE packets.

b. The Last Buffer Touched field is added to DATA/LDATA/NULL-ACK packets. This simplifies the data timer mechanism.

c. The KEEPALIVE packet is eliminated (its function is retained, using other packet types), and packet type numbers are changed.

d. Buffer numbers start with 1.

In addition, DATA packet checksumming is mandatory, and the transfer mode must be WRITE; that is, certain NETBLT options are disallowed in TACO2. Finally, half-duplex and simplex modes of operation are defined.

6.3 SLIP drivers.

The following C language functions send and receive SLIP packets. They depend on two functions, send_char() and recv_char(), which send and receive a single character over the serial line.

```
/* SLIP special character codes. These are OCTAL representations.
 */
#define END             0300    /* indicates end of packet */
#define ESC             0333    /* indicates byte stuffing */
#define ESC_END         0334     /* ESC ESC_END means END data byte */ #define
ESC_ESC         0335    /* ESC ESC_ESC means ESC data byte */
/* SEND_PACKET: sends a packet of length "len", starting at
  * location "p."
  */
void send_packet(p, len)
    char *p;
    int len; {

  /* send an initial END character to flush out any data that may
   * have accumulated in the receiver due to line noise
   */
     send_char(END);

  /* for each byte in the packet, send the appropriate character
   * sequence
   */
      while(len--) {
           switch(*p) {
          /* if it's the same code as an END character, we send
           * a special two character code so as not to make the
           * receiver think we sent an END
           */
           case END:
               send_char(ESC);
               send_char(ESC_END);
               break;

          /* if it's the same code as an ESC character,
           * we send a special two character code so as not
           * to make the receiver think we sent an ESC
           */
           case ESC:
               send_char(ESC);
               send_char(ESC_ESC);
               break;

          /* otherwise, we just send the character
          */
          default:
               send_char(*p);
               }
```

```
        p++;
        }

    /* tell the receiver that we're done sending the packet
     */
    send_char(END);
    }

/* RECV_PACKET: receives a packet into the buffer located at "p."
 *      If more than len bytes are received, the packet will
 *      be truncated.
 *      Returns the number of bytes stored in the buffer.
 */
int recv_packet(p, len)
    char *p;
    int len; {
    char c;
    int received = 0;

    /* sit in a loop reading bytes until we put together
     * a whole packet.
     * Make sure not to copy them into the packet if we
     * run out of room.
     */
    while(1) {
        /* get a character to process
        */
        c = recv_char();

         /* handle byte stuffing if necessary
          */
         switch(c) {

         /* if it's an END character then we're done with
          * the packet
          */
         case END:
             /* a minor optimization: if there is no
              * data in the packet, ignore it. This is
              * meant to avoid bothering IP with all
              * the empty packets generated by the
              * duplicate END characters which are in
              * turn sent to try to detect line noise.
              */
             if(received)
                 return received;
             else
                 break;

         /* if it's the same code as an ESC character, wait
          * and get another character and then figure out
          * what to store in the packet based on that.
          */
         case ESC:
```

```
            c = recv_char();

            /* if "c" is not one of these two, then we
             * have a protocol violation.  The best bet
             * seems to be to leave the byte alone and
             * just stuff it into the packet
             */
            switch(c) {
            case ESC_END:
                c = END;
                break;
            case ESC_ESC:
                c = ESC;
                break;
                }

        /* here we fall into the default handler and let
         * it store the character for us
         */
        default:
            if(received < len)
                p[received++] = c;
            }
        }
    }
```

6.4 Notes on FEC.

6.4.1 General notes on FEC. Numerous FEC codes exist, all with different properties, with the result that the proper matching of codes to channels is an important design aspect of any system that includes FEC. FEC may be implemented in hardware, firmware, or software, or by a combination of these methods. With respect to NITFS transmissions, FEC functions shall be realized by one of the following means: as a separate hardware device, called an FEC Applique; as hardware or firmware embedded in a system; or as an integral part of a modem (or in some cases, to a modem internal to a radio); or as host-resident software. The following sections treat each of these four possibilities separately. In each case, a survey of implementation relevant to NITFS is presented.

6.4.2 Discussion of FEC appliques. An FEC Applique is an "add-on" FEC device; a separately packaged piece of equipment whose primary function is to apply FEC encoding and decoding to a data stream. FEC Appliques represent perhaps the simplest method of adding FEC to an existing SIDS system; usually only correct cabling is required, plus adjustments as needed to account for any added delays in the FEC unit.

6.4.3 Discussion of FEC-I and FEC-II. At the low-to-moderate bit rates associated with tactical communications, it is practical to implement FEC encoding and decoding in host software. The advantage of software coding will, in general, not be as great as that provided by dedicated hardware, but the portability of standardized software FEC and its relatively low cost should lead to increased interoperability. The FEC-I and FEC-II codes described in 5.4.2.1 and appendix C are designed to be implementable entirely in software on a typical workstation or portable computer.

6.4.4 Interpretation of BERT results. The BERT test described in 5.4.2.3 measures a count of successfully received frames from a given number of attempts. For an HDLC channel with random error statistics, a count value of *N* provides an estimate of the random bit-error ratio as follows:

Standard BERT test:

$$BER = 1 - (N/1000)^{1/113}$$

Short BERT test:

$$BER = 1 - (N/200)^{1/113}$$

For a SLIP channel, no exact general formula ties BERT test results to Bit Error Ratio (BER) estimates. However, the above formulae still may be used to provide approximate guidance.

6.4.5 Performance considerations. To state the performance of an FEC code, assumptions must be made in two areas. First, the noise and errors associated with the channel must be characterized. Second, a metric for data integrity is needed.

An exhaustive survey of the possibilities for these two assumptions is beyond the scope of this document. Therefore, we will evaluate several of the subject FEC codes described in the previous section using the following very general assumptions.

Channel: We assume the channel exhibits random digital errors at various levels from 10^{-5} bit error ratio (BER) to a BER of several percent.

Performance metric: We use as a performance metric the relative data throughput for 152-byte datagrams, taking into account the following four factors:

a. HDLC framing, bit-stuffing, and CRC overhead,

b. FEC redundancy,

c. Packet loss due to errors uncorrectable by the FEC,

d. Packet loss due to unrecoverable HDLC framing errors.

For example, if there is no coding overhead, and also are no errors, the relative throughput by this measure is determined solely by the overhead of the HDLC framing, bit-stuffing, and CRC.

Based on these assumptions, figure 23 compares the performance of the following five coding systems. HDLC framing is assumed in each instance.

FEC-I	FEC-I as described in 5.4.1.1 of this document
FEC-II	FEC-II as described in appendix C of this document
HW-1/2	Rate 1/2 Reed-Solomon code, 6 bit symbols
HW-3/4	Rate 3/4 Reed-Solomon code, 6 bit symbols
V	Rate 1/2, constraint length 7, Viterbi Encoding
Z	No FEC

The data on figure 23 for FEC-I, HW-1/2, HW-3/4, V, and Z have been determined analytically and confirmed by experiment. The data for FEC-II are the result of analysis only.

As illustrated by figure 23, the six-bit Reed-Solomon codes (HW-1/2 and HW-3/4), and the Viterbi code (V), which have been implemented in hardware devices, outperform the software FEC specified in 5.4.2.1 and appendix C. This stems from placement of the Software FEC "above" the HDLC framing [as illustrated on figure 2, (c)], and the sensitivity of the HDLC framing mechanism to bit errors.

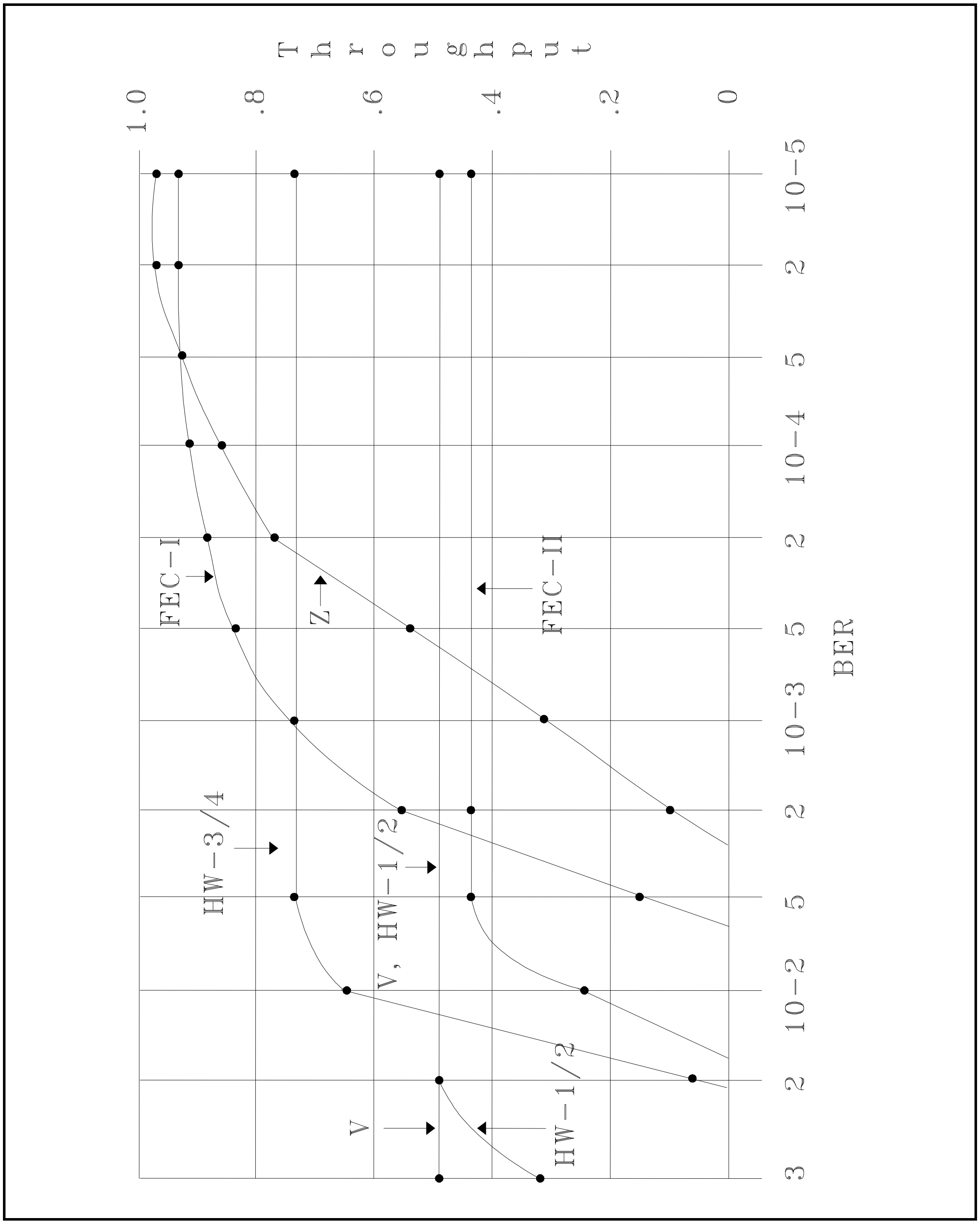

FIGURE 23. BER vs. relative throughput.

6.4.6 Selection of FEC coding options.

6.4.6.1 General discussion. Figure 23 provides some guidance in the selection of FEC coding. For random error channels, the BER may be measured by applying the BER test described in 5.4, or by other methods. Assume that the options available are no coding, FEC-I, and FEC-II. Based on this result and the data of figure 23, the no coding option is best for BER better than 5×10^{-5}, the FEC-II code preferred for BER worse than 3×10^{-3}, and the FEC-I code selected for BER between these two values.

Note that if the channel is bursty rather than random, conditions may exist in which the limited burst correction ability of FEC-I (33 bits) is exceeded, but the burst correction ability of FEC-II (790 bits) allows successful operation.

If a form of FEC is available as part of the specific datalink external to the SIDS system, the selection process becomes more complex. In many cases, the external system specific FEC will outperform the FEC-I and FEC-II codes and therefore is the first choice for routine operation of the specific SIDS system on the datalink. However, bypassing the system specific FEC and invoking FEC-I or FEC-II will allow interoperable communications between dissimilar SID systems on the same data link.

The following provides some general guidance as to the possible configurations of FEC-I and FEC-II FEC on various circuits.

6.4.6.2 Descriptions of circuits.

6.4.6.2.1 16 kbps UHF SATCOM. The circuit consists of Ultra High Frequency (UHF) transceivers communicating over satellite transponders, equipped with a KY-57 or Sunburst device operating at 16 kbps, using baseband modulation. This description also may be used for similar circuits including KY-57 encrypted line of sight (LOS) circuits using UHF or Very High Frequency (VHF) radios.

6.4.6.2.2 2.4 kbps UHF SATCOM. The circuits consist of UHF transceivers communicating over satellite transponders, equipped with a KG-84 or Sunburst device, and with internal BPSK modems operating at 2.4 kbps; or similar circuits using UHF or VHF radios operating LOS.

6.4.6.2.3 16 kbps UHF SATCOM with FEC applique. The circuit consists of UHF transceivers communicating over satellite transponders, equipped with a 16 kbps KY-57 or Sunburst device, and equipped with an FEC applique; or similar circuits using UHF or VHF radios operating LOS.

6.4.6.2.4 2.4 kbps UHF SATCOM with FEC applique. The circuits consist of UHF transceivers communicating over satellite transponders, equipped with a KG-84 or Sunburst device, and with internal Binary Phase Shift Keying (BPSK) modems operating at 2.4 kbps, and equipped with an FEC applique; or similar circuits using UHF or VHF radios operating LOS.

6.4.6.2.5 HF circuits. The circuit consists of High Frequency (HF) transceivers equipped with internal or external modems operating from 300 bps to 2400 bps.

6.4.6.2.6 HF circuits with hardware FEC. The circuit consists of HF transceivers equipped with internal or external modems operating from 300 bps to 2400 bps, with the addition of FEC either internal to the modem or by an FEC applique.

6.4.6.2.7 TRI-TAC. 16 or 32 kbps Tri-Service Tactical Communications (TRI-TAC) connection, KY-68 encrypted.

6.4.6.2.8 Telephone circuit. Standard telephone circuit equipped with a modem that does not included retransmission-based error control such as a Bell 212/224, V.26, V.27, or V.32 modem.

6.4.6.2.9 Telephone circuit with error control. Standard telephone circuit equipped with a modem that implements retransmission protocols such as MNP4/MNP5, V.42, or PEP.

6.4.6.2.10 DAMA. Viterbi-encoded U.S. Navy Demand Assignment Multiple Access (DAMA) circuit.

6.4.6.3 Recommended modes. Table II shows the recommended configuration of FEC-I and FEC-II as a function of the above circuit descriptions. If the communication equipment includes FEC, use of FEC-I or FEC-II is not recommended.

TABLE II. Recommended modes.

Circuit Type	Recommended Modes
6.4.6.2.1	FEC-II
6.4.6.2.2	Unencoded, FEC-I, FEC-II
6.4.6.2.3	Unencoded, FEC-I, FEC-II
6.4.6.2.4	Unencoded, FEC-I, FEC-II
6.4.6.2.5	FEC-II
6.4.6.2.6	FEC-I, FEC-II
6.4.6.2.7	FEC-I
6.4.6.2.8	FEC-I, FEC-II
6.4.6.2.9	Unencoded
6.4.6.2.10	Unencoded

6.5 Effectivity summary. Some of the capabilities specified in this document are not required as of the issue date of the document. All such capabilities are marked with effectivity numbers, for example, (Effectivity 1). Each effectivity number will be replaced by a specific date in subsequent releases of this document.

6.5.1 Effectivity 1 - FEC I and Bit Error Ratio Test (BERT).

a. 4.1.6 FEC. Forward Error Correction (FEC) is a mandatory component of the TACO2 protocol stack whose use in a particular circuit is user selectable (Effectivity 1). …

6.5.2 Effectivity 2 - FEC II.

a. APPENDIX C FEC-II CODE. (The contents of this section are (Effectivity 2) pending further implementation and testing of the proposed FEC code.)

b. 5.4.2.2.3 FEC-II. FEC-II is applied to a SLIP and/or HDLC encapsulated datalink as described in appendix C (Effectivity 2).

6.5.3 Effectivity 3 - Header abbreviation and client-controlled flow.

a. 4.1.5 Header Abbreviation sublayer. TACO2 provides a mechanism for header abbreviation across point-to-point links. Use of the header abbreviation sublayer is optional: its inclusion in any compliant implementation of TACO2 shall be mandatory (Effectivity 3).

b. 5.2.3.5 Client-controlled flow. (Effectivity 3)

c. 5.4.1 Header Abbreviation sublayer. TACO2 provides a mechanism for header abbreviation across point-to-point links. Use of the header abbreviation sublayer is optional: its inclusion in any compliant implementation of TACO2 shall be mandatory (Effectivity 3).

6.5.4 Effectivity 4 - Pull vs. push.

a. 5.1 NITFS reliable transfer server for TACO2 (TACO2 NRTS). The TACO2 NRTS described here assumes an active sender and a passive receiver ("push" operation); as of the effectivity date (Effectivity 4) the TACO2 NRTS shall also support an active receiver and passive sender ("pull" operation).

b. 5.2.9.2.5 Direction. (Effectivity 4) Until the effectivity date, operation of TACO2 is defined only for "M" set to 1; that is, TACO2 allows only active sending and passive receiving. Following that date, operation with "M" set to 0 is also permissible.

6.5.5 Effectivity 5 - Multicast.

a. 5.3.1.1 IP augmentations. … TACO2 supports a limited form of multicasting by allowing simplex receivers to "listen in" on simplex, half-duplex, or full-duplex transmissions; (Effectivity 5: later versions of TACO2 may support acknowledged multicast).

6.5.6 Effectivity 6 - Medium Access Control layer.

a. 5.4 Data link layer. The Data Link layer in TACO2 is divided into three sublayers: Header Abbreviation, FEC, and Framing. (Effectivity 6: a Medium Access Control Layer, just below the Framing Sublayer, is under consideration.)

6.5.7 Effectivity 8 - Defense Information Systems Network (DISN).

a. DISA/JIEO Circular 9008

b. DISA/JIEO Specification 9137

c. DISA/JIEO Specification 9138

d. DISA/JIEO Specification 9139

e. DISA/JIEO Specification 9140

6.7 Subject term (key word) listing.

Error detection
Forward error correction (FEC)
Frames
HDLC
ICMP
IP
Message Transfer Facility
NETBLT
Packets
Secondary Imagery Dissemination Systems
SIDS
SLIP

APPENDIX A

FINITE FIELD ARITHMETIC FOR FEC-I AND FEC-II CODES

10. Scope. This appendix is not a mandatory part of the standard. The information contained in it is intended for guidance only.

20. Applicable documents. This section is not applicable to this appendix.

30. Finite field arithmetic for FEC-I and FEC-II codes. The field arithmetic used in these codes is defined by the field generator polynomial $x^8 + x^5 + x^3 + x^2 + 1$. This appendix describes what this means in terms of performing field operations such as addition and multiplication as part of a computation implementing the FEC codes.

There are 256 field elements, each represented by an eight-bit byte. The following elements (given in hexadecimal) have special significance:

0x00 --- 0, the additive identity
0x01 --- 1, the multiplicative identity
0x02 --- α, the field generator

Arithmetic computation for this field can be defined by the following six postulates:

a. For any elements x and y, $x+y$ is the bitwise exclusive-OR of x and y. Therefore, $x + 0 = 0 + x = x$; $x + x = 0$; $x + y = y + x$; and for elements x, y and z, $(x+y) + z = x + (y+z)$.

b. For any element x, $x \times 1 = x$.

c. Commutative law for multiplication: for any elements x and y, $xy = yx$.

d. Associative law for multiplication: for any elements x, y and z, $(xy)z = x(yz)$.

e. Distributive law: for any elements c, x, and y, $c(x+y) = cx + cy$.

f. The following multiplication table, combined with the distributive law, defines multiplication for the entire field (all values are hexadecimal):

	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
0x01	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
0x02	0x02	0x04	0x08	0x10	0x20	0x40	0x80	0x2D
0x04	0x04	0x08	0x10	0x20	0x40	0x80	0x2D	0x5A
0x08	0x08	0x10	0x20	0x40	0x80	0x2D	0x5A	0xB4
0x10	0x10	0x20	0x40	0x80	0x2D	0x5A	0xB4	0x45
0x20	0x20	0x40	0x80	0x2D	0x5A	0xB4	0x45	0x8A
0x40	0x40	0x80	0x2D	0x5A	0xB4	0x45	0x8A	0x39
0x80	0x80	0x2D	0x5A	0xB4	0x45	0x8A	0x39	0x72

Based on the above, exponentiation of a field element x by a non-negative integer n is defined so that x^0 is 1, and x^n is the result of multiplying x together n times. The field element α is said to generate the field because α^0 through α^{254} are distinct, and are the 255 non-zero field elements.

The inverse of a field element α^i is $\alpha^{(255-i)}$: $\alpha^i \alpha^{255-i} = 1$ for any integer i between 0 and 254. Exponentiation by negative integer exponents is defined such that x^{-1} is the inverse of x.

Example: multiply 0x80 by 0x06, using the multiplication table and the distributive law:

0x80 x 0x06
= 0x80(0x04 + 0x02)
= 0x80 x 0x04 + 0x80 x 0x02
= 0x5A + 0x2D
= 0x77.

Example: compute the value of $\alpha^3 + \alpha^{65}$

α^3 = 0x08
α^{65} = 0xC1
$\alpha^3 + \alpha^{65}$ = 0xC9

The 256 field elements are provided in Table A-1.

Table A-I. Field Elements.

i	a^i	i	a^i	i	a^i	i	a^i	i	a^i
0	01	52	95	104	4E	156	F5	208	DE
1	02	53	07	105	9C	157	C7	209	91
2	04	54	0E	106	15	158	A3	210	0F
3	08	55	1C	107	2A	159	6B	211	1E
4	10	56	38	108	54	160	D6	212	3C
5	20	57	70	109	A8	161	81	213	78
6	40	58	E0	110	7D	162	2F	214	F0
7	80	59	ED	111	FA	163	5E	215	CD
8	2D	60	F7	112	D9	164	BC	216	B7
9	5A	61	C3	113	9F	165	55	217	43
10	B4	62	AB	114	13	166	AA	218	86
11	45	63	7B	115	26	167	79	219	21
12	8A	64	F6	116	4C	168	F2	220	42
13	39	65	C1	117	98	169	C9	221	84
14	72	66	AF	118	1D	170	BF	222	25
15	E4	67	73	119	3A	171	53	223	4A
16	E5	68	E6	120	74	172	A6	224	94
17	E7	69	E1	121	E8	173	61	225	05
18	E3	70	EF	122	FD	174	C2	226	0A

i	a^i
19	EB
20	FB
21	DB
22	9B
23	1B
24	36
25	6C
26	D8
27	9D
28	17
29	2E
30	5C
31	B8
32	5D
33	BA
34	59
35	B2
36	49
37	92
38	09
39	12

i	a^i
71	F3
72	CB
73	BB
74	5B
75	B6
76	41
77	82
78	29
79	52
80	A4
81	65
82	CA
83	B9
84	5F
85	BE
86	51
87	A2
88	69
89	D2
90	89
91	3F

i	a^i
123	D7
124	83
125	2B
126	56
127	AC
128	75
129	EA
130	F9
131	DF
132	93
133	0B
134	16
135	2C
136	58
137	B0
138	4D
139	9A
140	19
141	32
142	64
143	C8

i	a^i
175	A9
176	7F
177	FE
178	D1
179	8F
180	33
181	66
182	CC
183	B5
184	47
185	8E
186	31
187	62
188	C4
189	A5
190	67
191	CE
192	B1
193	4F
194	9E
195	11

i	a^i
227	14
228	28
229	50
230	A0
231	6D
232	DA
233	99
234	1F
235	3E
236	7C
237	F8
238	DD
239	97
240	03
241	06
242	0C
243	18
244	30
245	60
246	C0
247	AD

TABLE A-I. Field Elements - Continued.

i	a^i
40	24
41	48
42	90
43	0D
44	1A
45	34
46	68
47	D0
48	8D
49	37
50	6E
51	DC

i	a^i
92	7E
93	FC
94	D5
95	87
96	23
97	46
98	8C
99	35
100	6A
101	D4
102	85
103	27

i	a^i
144	BD
145	57
146	AE
147	71
148	E2
149	E9
150	FF
151	D3
152	8B
153	3B
154	76
155	EC

i	a^i
196	22
197	44
198	88
199	3D
200	7A
201	F4
202	C5
203	A7
204	63
205	C6
206	A1
207	6F

i	a^i
248	77
249	EE
250	F1
251	CF
252	B3
253	4B
254	96
255	00

APPENDIX B

EXAMPLE SOFTWARE FOR FEC-I

10. Scope. This appendix is not a mandatory part of the standard. The information contained in it is intended for guidance only.

20. Applicable documents. This section is not applicable to this appendix.

30. Example software for FEC-I.

Contents of the file "ReadMe" are as follows:

a. This directory contains three files:

- this file ("ReadMe")
- "x3b11", which has log/alog tables for the finite field
- sample.c, which contains sample encode() and decode() functions for the FEC-I code

When compiled and executed, sample.c performs a sample decoding and should produce the following output:

```
remainder: 33
remainder: 127
remainder: 74
remainder: 208
remainder: 239
remainder: 95
remainder: 43
remainder: 181
remainder: 41
remainder: 150
degW 5
d: 0

dtmp at end of chien(): 0

length: 152
degree (should be 5): 5
uncorrected errors (should be 0): 0
```

Contents of the file "x3b11" are as follows:

```
256

00 00 01 f0 02 e1 f1 35 03 26 e2 85 f2 2b 36 d2
04 c3 27 72 e3 6a 86 1c f3 8c 2c 17 37 76 d3 ea
05 db c4 60 28 de 73 67 e4 4e 6b 7d 87 08 1d a2
f4 ba 8d b4 2d 63 18 31 38 0d 77 99 d4 c7 eb 5b
06 4c dc d9 c5 0b 61 b8 29 24 df fd 74 8a 68 c1
e5 56 4f ab 6c a5 7e 91 88 22 09 4a 1e 20 a3 54
f5 ad bb cc 8e 51 b5 be 2e 58 64 9f 19 e7 32 cf
39 93 0e 43 78 80 9a f8 d5 a7 c8 3f ec 6e 5c b0
07 a1 4d 7c dd 66 da 5f c6 5a 0c 98 62 30 b9 b3
2a d1 25 84 e0 34 fe ef 75 e9 8b 16 69 1b c2 71
e6 ce 57 9e 50 bd ac cb 6d af a6 3e 7f f7 92 42
89 c0 23 fc 0a b7 4b d8 1f 53 21 49 a4 90 55 aa
f6 41 ae 3d bc ca cd 9d 8f a9 52 48 b6 d7 bf fb
2f b2 59 97 65 5e a0 7b 1a 70 e8 15 33 ee d0 83
3a 45 94 12 0f 10 44 11 79 95 81 13 9b 3b f9 46
d6 fa a8 47 c9 9c 40 3c ed 82 6f 14 5d 7a b1 96

01 02 04 08 10 20 40 80 2d 5a b4 45 8a 39 72 e4
e5 e7 e3 eb fb db 9b 1b 36 6c d8 9d 17 2e 5c b8
5d ba 59 b2 49 92 09 12 24 48 90 0d 1a 34 68 d0
8d 37 6e dc 95 07 0e 1c 38 70 e0 ed f7 c3 ab 7b
f6 c1 af 73 e6 e1 ef f3 cb bb 5b b6 41 82 29 52
a4 65 ca b9 5f be 51 a2 69 d2 89 3f 7e fc d5 87
23 46 8c 35 6a d4 85 27 4e 9c 15 2a 54 a8 7d fa
d9 9f 13 26 4c 98 1d 3a 74 e8 fd d7 83 2b 56 ac
75 ea f9 df 93 0b 16 2c 58 b0 4d 9a 19 32 64 c8
bd 57 ae 71 e2 e9 ff d3 8b 3b 76 ec f5 c7 a3 6b
d6 81 2f 5e bc 55 aa 79 f2 c9 bf 53 a6 61 c2 a9
7f fe d1 8f 33 66 cc b5 47 8e 31 62 c4 a5 67 ce
b1 4f 9e 11 22 44 88 3d 7a f4 c5 a7 63 c6 a1 6f
de 91 0f 1e 3c 78 f0 cd b7 43 86 21 42 84 25 4a
94 05 0a 14 28 50 a0 6d da 99 1f 3e 7c f8 dd 97
03 06 0c 18 30 60 c0 ad 77 ee f1 cf b3 4b 96 00
```

Contents of the file "sample.c" are as follows:

```
* sample version of FEC-I encode and decode routines --
   functional for unencoded packet lengths up to 152 bytes */

#include <stdio.h>
#define K 152
#define GFCALCFILE "x3b11"

/* array of constants used by Welch-Berlekamp iteration */

unsigned char x[10][12] = {

{ 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
{ 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x2d, 0x5a, 0xb4, 0x45 },
{ 0x01, 0x04, 0x10, 0x40, 0x2d, 0xb4, 0x8a, 0x72, 0xe5, 0xe3, 0xfb, 0x9b },
{ 0x01, 0x08, 0x40, 0x5a, 0x8a, 0xe4, 0xe3, 0xdb, 0x36, 0x9d, 0x5c, 0xba },
{ 0x01, 0x10, 0x2d, 0x8a, 0xe5, 0xfb, 0x36, 0x17, 0x5d, 0x49, 0x24, 0x1a },
{ 0x01, 0x20, 0xb4, 0xe4, 0xfb, 0x6c, 0x5c, 0xb2, 0x24, 0x34, 0x6e, 0x1c },
{ 0x01, 0x40, 0x8a, 0xe3, 0x36, 0x5c, 0x49, 0x90, 0x8d, 0x0e, 0xf7, 0xaf },
{ 0x01, 0x80, 0x72, 0xdb, 0x17, 0xb2, 0x90, 0x37, 0x38, 0x7b, 0xef, 0x82 },
{ 0x01, 0x2d, 0xe5, 0x36, 0x5d, 0x24, 0x8d, 0x38, 0xf6, 0xcb, 0xa4, 0x69 },
{ 0x01, 0x5a, 0xe3, 0x9d, 0x49, 0x34, 0x0e, 0x7b, 0xcb, 0x65, 0x89, 0x35 }
};


/* packet structure -- maximum lengths:
      netblt packet data field: 100 bytes
      ip datagram: 152 bytes
      encoded packet: 162 bytes (1 RS codeword)
     structures sized for 700 byte packets - later */

struct packet {
     short length;
     unsigned char data[802];
     };

unsigned short log[256], alog[256];
unsigned char w[6], n[6], m[6], v[6];
unsigned char loc[6], val[6];
short NumErrs;
short d,degW;
unsigned char a,b,s;

/* main() first initializes log and alog tables */

main()
{
struct packet pack;
int x,i;
scan_gfcalc();  /* function to read in log, alog tables */

/* test code creating a packet with five errors */

pack.length = 162;
for (i=0;i<162;i++) {
     pack.data[i] = (i>4) ? 0 : I+1;
```

```
     }

/* decode the packet */

decode(&pack);

/* now print some more stuff to verify successful decode */

printf("length: %d\n",pack.length);
d=0;
for (i=0;i<152;i++) if (pack.data[i]) d++;
printf("degree (should be 5): %d\n",degW);
printf("uncorrected errors (should be 0): %d\n",d);

}

scan_gfcalc()
{
FILE *fp; int i,j;
if ((fp=fopen(GFCALCFILE,"r"))==NULL)
     {(void)fprintf(stderr,"bad file\n"); exit(1);}
(void) fscanf(fp,"%d",&j); /* dummy scan */
/* printf("%d\n",j); */

for (i=0;i<256;i++)
     { (void) fscanf(fp,"%x",&j);
         /* printf("%d\n",j); */
       log[i] = (unsigned short) j; }
for (i=0;i<256;i++)
     { (void) fscanf(fp,"%x",&j); alog[i] = (unsigned short) j; }
(void) fclose(fp);
}

unsigned short
invert(x)
unsigned short x;
{
return(alog[(255 - log[x]) % 255]);
}

unsigned short
mult(x,y)
unsigned short x,y;
{
return(x && y ? alog[(log[x] + log[y]) % 255] : 0);
}

/* function to encode a single packet */

encode(p)
struct packet *p;
{
int l,i,j;
if (!(l = p->length)) return;
if (l>152) return; /* later -- encode longer packets */

/* initialize check bytes to zero */

for (j=0;j<10;j++) p->data[l+j] = 0;
```

```
/* encode by computing each check byte.  i and j have the same
   definition as in the equation defining the check bytes in 5.1 */

for (j=0;j<10;j++)
     for (i=10;i<l+10;i++)
          p->data[l+j] ^= mult(p->data[l+9-i],invert(alog[i]^alog[j]));

/* increase packet length by 10 */

p->length += 10;
}

/* function to decode a single packet: first re-encode, then
   cal wb(), chien(), correct() */

decode(p)
struct packet *p;
{
int l,i,j;
if (!(l = (p->length)-10)) return;
if (l>152) return; /* later -- decode longer packets */

/* re-encode each check byte -- this is the same as encoding, but
   without zeroing the check bytes first. i and j have the same
   definition as in the equation defining encoding checks in 5.1 */

for (j=0;j<10;j++)
     for (i=10;i<l+10;i++)
          p->data[l+j] ^= mult(p->data[l+9-i],invert(alog[i]^alog[j]));

/* done re-encoding, proceed with decode */

wb(p);
printf("degW %d\n",degW);
if (chien(p)) correct(p);
p->length -= 10;
}


/* wb() performs the Welch-Berlekamp iteration */

wb(p)
struct packet *p;
{

/* initialize welch-berlekamp */

register short j,k,lim;

d=0;
for (j=0;j<6;j++) w[j]=m[j]=n[j]=v[j]=0;
w[0]=m[0]=1;

/* welch-berlekamp iteration */

for (j=0;j<10;j++) {

    /* set for-loop limit for polynomial operations */
    lim = ((j>4) ? 5 : j+1);
```

```
/* get remainder from packet */

s = p->data[(p->length - 10) + j];
printf("remainder: %d\n",s);

/* evaluate a and b */

a = b = 0;
for (k=0;k<=lim;k++) {
      a ^= mult(x[j][k],n[k]) ^ mult(s,mult(x[j][k],w[k]));
      b ^= mult(x[j][k],m[k]) ^ mult(s,mult(x[j][k],v[k]));
      }

/* test d, a, b and do a 4-way branch */

if ((d<=0) &&  !a) {

     /* update M, V */
     for (k=lim;k>0;k--) {
           m[k] = m[k-1] ^ mult(x[j][1],m[k]);
           v[k] = v[k-1] ^ mult(x[j][1],v[k]);
           }
     m[0] = mult(x[j][1],m[0]);
     v[0] = mult(x[j][1],v[0]);

     d--;
     continue;
     }

if ((d>0) &&  !b) {

     /* update N, W */
     for (k=lim;k>0;k--) {
           n[k] = n[k-1] ^ mult(x[j][1],n[k]);
           w[k] = w[k-1] ^ mult(x[j][1],w[k]);
           }
     n[0] = mult(x[j][1],n[0]);
     w[0] = mult(x[j][1],w[0]);

     d++;
     continue;
     }

if ((d<=0) && a) {

     /* update M, V */
     for (k=0;k<=lim;k++) {
           m[k] = mult(a,m[k]) ^ mult(b,n[k]);
           v[k] = mult(a,v[k]) ^ mult(b,w[k]);
           }

     /* update N, W */
     for (k=lim;k>0;k--) {
           n[k] = n[k-1] ^ mult(x[j][1],n[k]);
           w[k] = w[k-1] ^ mult(x[j][1],w[k]);
           }
     n[0] = mult(x[j][1],n[0]);
     w[0] = mult(x[j][1],w[0]);
```

```
            d++;
            continue;
            }

      if ((d>0) && b) {

            /* update N, W */
            for (k=0;k<=lim;k++) {
                  n[k] = mult(b,n[k]) ^ mult(a,m[k]);
                  w[k] = mult(b,w[k]) ^ mult(a,v[k]);
                  }

            /* update M, V */
            for (k=lim;k>0;k--) {
                  m[k] = m[k-1] ^ mult(x[j][1],m[k]);
                  v[k] = v[k-1] ^ mult(x[j][1],v[k]);
                  }
            m[0] = mult(x[j][1],m[0]);
            v[0] = mult(x[j][1],v[0]);

            d--;
            continue;
            }

      }

/* done with W-B iteration; must set degW to degree of w[] */

degW=6;
while (!w[(degW--)-1]);
}

/* value() is called by chien with an argument which is the
   alog of a message error location.  It returns the error value
   N(alog[l]) / W'(alog[l]) */

unsigned char
value(l)
short l;
{
unsigned char num,den,t,a;
t = (a = alog[l]);
num = n[0];
num ^= mult(n[1],t);
t = mult(t,a);
num ^= mult(n[2],t);
t = mult(t,a);
num ^= mult(n[3],t);
t = mult(t,a);
num ^= mult(n[4],t);
den = w[1];
t = mult(a,a);
den ^= mult(w[3],t);
t = mult(t,t);
den ^= mult(w[5],t);
return(mult(num,invert(den)));
}

/* chien searches for roots of w, updating the arrays
```

```
   loc[] and val[].  chien returns 1 if decodable, in
   which case loc[] and val[] have NumErrs location/value
   pairs; chien returns 0 if undecodable */

chien(p)
struct packet *p;
{
unsigned short sum,w0,w1,w2,w3,w4,w5;
short l,lim;
short dtmp,numerrs;

printf("d: %d\n\n",d);

if (d>0) return(0); /* wb exited undecodable */

dtmp = degW;

/* first search for check roots of w, starting by initializing
   w0, w1, w2, w3, w4, w5 for location alog[-1] = alog[254].  When a
   root is found we only decrement dtmp */

#define A254_1   0x96
#define A254_2   0x4b
#define A254_3   0xb3
#define A254_4   0xcf
#define A254_5   0xf1

w0 = w[0];
w1 = mult(w[1],A254_1);
w2 = mult(w[2],A254_2);
w3 = mult(w[3],A254_3);
w4 = mult(w[4],A254_4);
w5 = mult(w[5],A254_5);

for (l=0;l<10;l++) {
    sum = w0;
    sum ^= (w1 = mult(w1,alog[1]));
    sum ^= (w2 = mult(w2,alog[2]));
    sum ^= (w3 = mult(w3,alog[3]));
    sum ^= (w4 = mult(w4,alog[4]));
    sum ^= (w5 = mult(w5,alog[5]));
    if (!sum) dtmp--;
    }

/* printf("degW less number of check errors: %d\n\n",dtmp); */

/* w0...w5 are now set for location alog[9], and we are
   ready to search for message roots starting at the end
   of the message */

lim = (p->length);
NumErrs = 0;

for (l=10;l<lim;l++) {
         sum = w0;
         sum ^= (w1 = mult(w1,alog[1]));
         sum ^= (w2 = mult(w2,alog[2]));
         sum ^= (w3 = mult(w3,alog[3]));
         sum ^= (w4 = mult(w4,alog[4]));
```

```
         sum ^= (w5 = mult(w5,alog[5]));
    if (!sum) {
         dtmp--;
         loc[NumErrs] = l;
         val[NumErrs++] = value(l);
         }
    }

printf("dtmp at end of chien(): %d\n\n",dtmp);

return(dtmp == 0);
}
/* correct() fixes the errors in the message section of the
   packet */


correct(p)
struct packet *p;
{
int i;
for (i=0;i<NumErrs;i++)
     p->data[(p->length) - loc[i] - 1]  ^= val[i];
}
```

APPENDIX C

FEC-II CODE

(The contents of this section are (Effectivity 2) pending further implementation and testing of the proposed FEC code.)

10. Scope. This appendix is not a mandatory part of the standard. The information it contains is intended for guidance only.

20. Applicable documents. This section is not applicable to this appendix.

30. FEC-II Code. FEC-II encoding applies both Reed-Solomon and BCH coding for operation in high error environments. Further, FEC-II encodes a single datagram or section of a datagram into a group of "fragments," or small packets of 12 bytes each. The BCH coding protects each fragment, and using the Reed-Solomon coding, up to eight fragments may be lost in transmission while still allowing the receiver to recover the datagrams.

(The term fragment used in this section, is unrelated to the fragments defined by the Internet Protocol.)

The individual fragments shall be encapsulated by the data link layer, generally either as a SLIP frame as specified in 5.4.3.2, or as an HDLC frame as specified in 5.4.3.1. If a FEC-II fragment is encapsulated as an HDLC frame, the 2-byte HDLC frame opening sequence, as well as the 2-byte CRC, shall not be included in the frame.

Datagrams up to length 382 bytes may be encoded by the FEC-II coding process. The first step shall be to represent the datagram by either one or two "sections." If the datagram is of length L, where L is 191 bytes or less, it shall be represented by a single "section" containing L + one bytes as follows: an initial byte containing the byte count L, followed by the L bytes contained in the datagram. If the datagram is of length L, where L is 192 through 382 bytes inclusive, it shall be represented by two sections. The first section shall consist of an initial byte with value 255 (decimal), followed by the first 191 bytes contained in the datagram. The second section shall consist of an initial byte with value (L - 191), followed by the remaining bytes of the datagram.

First we will specify the format of a single fragment, and then describe how the input bytes used to form the fragments are derived from the unencoded section.

Each 12-byte fragment is formed from an 8-byte sub-block, which is made up of the "input bytes" on figure C-1.

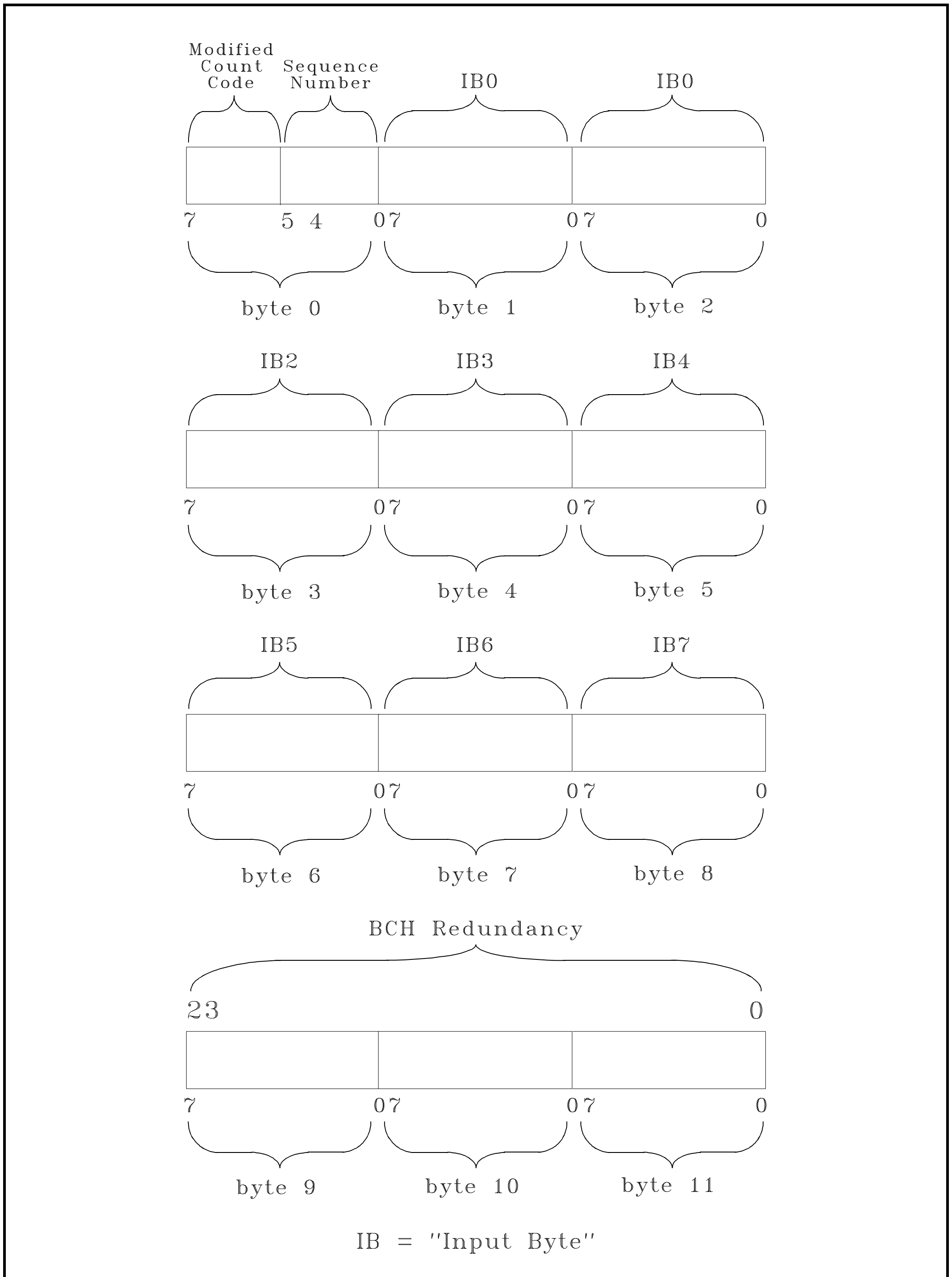

FIGURE C-1. FEC-II fragment format.

The origin of the 3-bit Modified Count Code and 5-bit sequence number will be described shortly. The 24-bit BCH redundancy field is formed using the following polynomial:

$$x^{24} + x^{23} + x^{21} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{13} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + 1$$

To calculate the BCH redundancy, feed the following bits, in the order shown, into a feedback shift register, initialized with the hexadecimal value 0x0000FF, and wired according to the above polynomial:

Modified Count Code bits 2 through 0
Sequence Number bits 4 through 0
IB0 bits 7 through 0
IB1 bits 7 through 0
.
.
.
IB7 bits 7 through 0

The above describes how to form a fragment given the input bytes, modified count code, and sequence number. We now describe how the input bytes, modified count codes, and sequence numbers are derived.

First, the unencoded section is split into a sequence of 8-byte sub-blocks. Eight more 8-byte sub-blocks containing Reed-Solomon redundancy are then created. Each sub-block contains bytes numbered 0 through 7.

Eight choices exist for the number of message sub-blocks used to represent a datagram, as determined by a 3-bit count code in table C-I.

TABLE C-I. Count codes for different length sections.

Count Code	Number of Message Sub-blocks	Sequence Numbers of Message Sub-blocks	Nominal Section Length (bytes)
0	4	0-3	32
1	6	0-5	48
2	8	0-7	64
3	10	0-9	80
4	13	0-12	104
5	16	0-15	128
6	19	0-18	152
7	24	0-23	192

The eight Reed-Solomon Redundancy Sub-blocks always have sequence numbers 24 through 31.

The 3-bit Modified Count Code inserted in the fragment is the bitwise-exclusive-OR formed from the above Count Code, and a residue computed from the remaining data in the fragment. This residue is defined as follows (using binary arithmetic):

$$Residue = \left\{ (sequence\ number \bmod 9) + \sum_{i=0}^{7} (Input\ byte\ i) \bmod 9 \right\} \bmod 8$$

Each byte in a Reed-Solomon Redundancy Sub-block with sequence number j is computed from the correspondingly-numbered bytes M_i in each of the message sub-blocks as follows:

$$C_{T(j)} = \sum_i \frac{M_i}{a^{T(i)} = a^{T(j)}} \quad , \quad 24 \le j \le 31$$

where i ranges over the message sub-block sequence numbers, and the correspondence between sequence numbers and locations within the Reed-Solomon codeword is:

x	T[x]
24	0
25	1
26	2
27	3
28	4
29	5
30	6
31	7
0	10
1	11
2	12
.	.
.	.
.	.
23	33

(The expression above uses Galois Field arithmetic as described in 5.4.2.1 for the FEC-I code.)

Presently, FEC-II encoding is not specified for datagrams whose unencoded length is greater than 382 bytes. Should a FEC-II encoder be presented with such a datagram, the correct action is to transmit it without any encoding.

Because the FEC-II encoding process has the effect of adding zero-fill to a datagram to achieve a standard length, a receiving system must determine the actual length of the original datagram. It does this by examining the length field in the Internet Protocol (IP) header, or, in the case where header abbreviation is used, the length field in the abbreviated header.

CONCLUDING MATERIAL

Custodians:
- Army - SC
- Navy - SC
- Air Force - 90
- Misc - DC

Review activities:
- OASD - IR
- Army - AC, PT
- Navy - CH, EC, MC, ND, NO, OM, TD
- Air Force - 02, 17, 90
- DLA - DH
- Misc - NS, MP, DI, DC, 29

User activities:
- Army - CR, PT
- Navy - MC, EC, TD
- Air Force - 13
- Misc - MP, DI, 29

Civil agency coordinating activities:
- USDA - AFS, APS
- COM - NIST
- DOE
- EPA
- GPO
- HHS - NIH
- DOI - BLM, GES, MIN
- DOT - CGCT

Preparing activity:
- Misc - DC

Agent:
- Not applicable

(Project DCPS-008)

STANDARDIZATION DOCUMENT IMPROVEMENT PROPOSAL

INSTRUCTIONS

1. The preparing activity must complete blocks 1,2, 3, and 8. In block 1, both the document number and revision letter should be given.
2. The submitter of this form must complete blocks 4, 5, 6, and 7.
3. The preparing activity must provide a reply within 30 days from receipt of the form.

NOTE: This form may not be used to request copies of documents, nor to request waivers, or clarification of requirements on current contracts. Comments submitted on this form do not constitute or imply authorization to waive any portion of the referenced document(s) or to amend contractual requirements.

I RECOMMEND A CHANGE:	1. DOCUMENT NUMBER **MIL-STD-2045-44500**	2. DOCUMENT DATE (YYMMDD) **930618**

3. DOCUMENT TITLE **TACTICAL COMMUNICATIONS PROTOCOL 2 (TACO2) for the NITFS**

4. NATURE OF CHANGE *(Identify paragraph number and include proposed rewrite, if possible. Attach extra sheets as needed.)*

5. REASON FOR RECOMMENDATION

6. SUBMITTER

a. NAME *(Last, First, Middle Initial)*	b. ORGANIZATION	
c. ADDRESS *(Include Zip Code)*	d. TELEPHONE *(Include Area Code)* (1) Commercial (2) AUTOVON *(If applicable)*	7. DATE SUBMITTED *(YYMMDD)*

8. PREPARING ACTIVITY **DEFENSE INFORMATION SYSTEMS AGENCY**

a. NAME **DISA/JIEO/CFS/TBBD**	b. TELEPHONE *(Include Area Code)* (1) Commercial (2) AUTOVON
c. ADDRESS *(Include Zip Code)* **Fort Monmouth, NJ 07703-5613**	**IF YOU DO NOT RECEIVE A REPLY WITHIN 45 DAYS, CONTACT:** Defense Quality and Standardization Office 5203 Leesburg Pike, Suite 1403, Falls Church, VA 22041-3466 Telephone (703) 756-2340 AUTOVON 289-2340